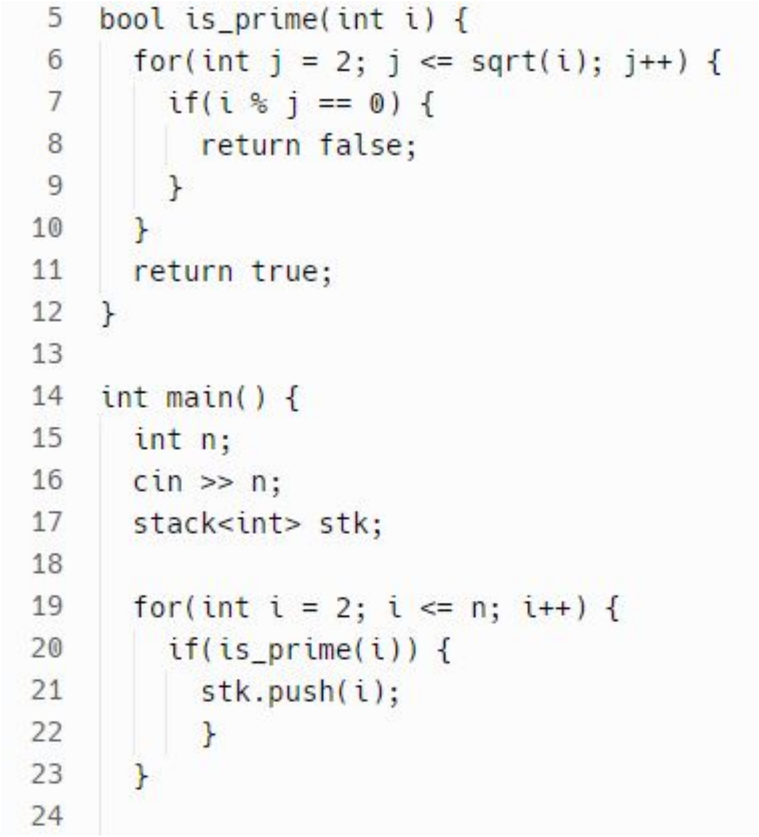
Class related - Survey results

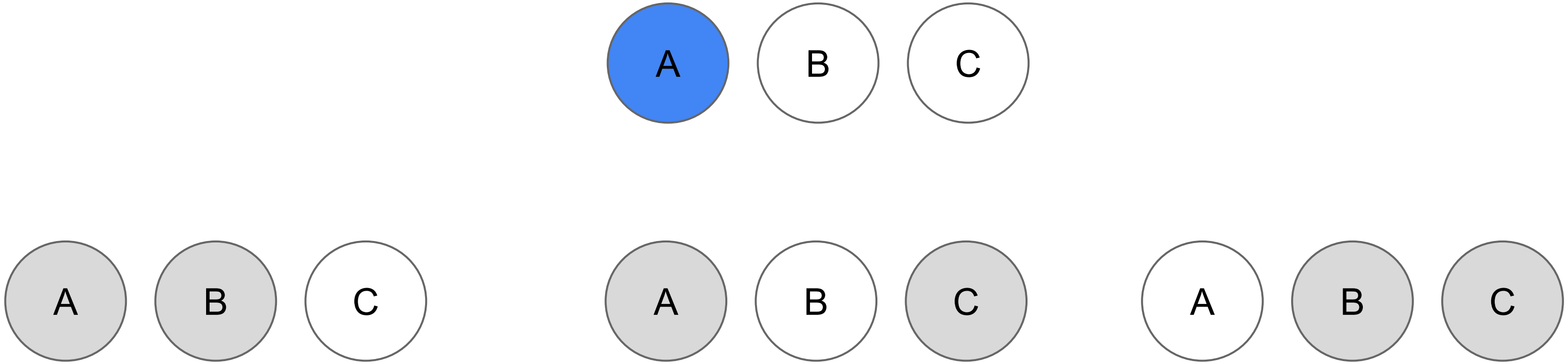
- Class difficulty level: 3 ~ 4
- Homework difficulty level: 90% moderate, 2~4 hours
- Solutions to homework problems
- Homework topics
  - Related to new concepts
  - Related to old concepts we covered
  - Coding practice
- Teacher/TA emails:
  - Nick: nick@x-camp.academy, class related questions
  - Peter: peter@x-camp.academy   Cory: coryf615@gmail.com, homework related questions

Homework - Prime List - P9495

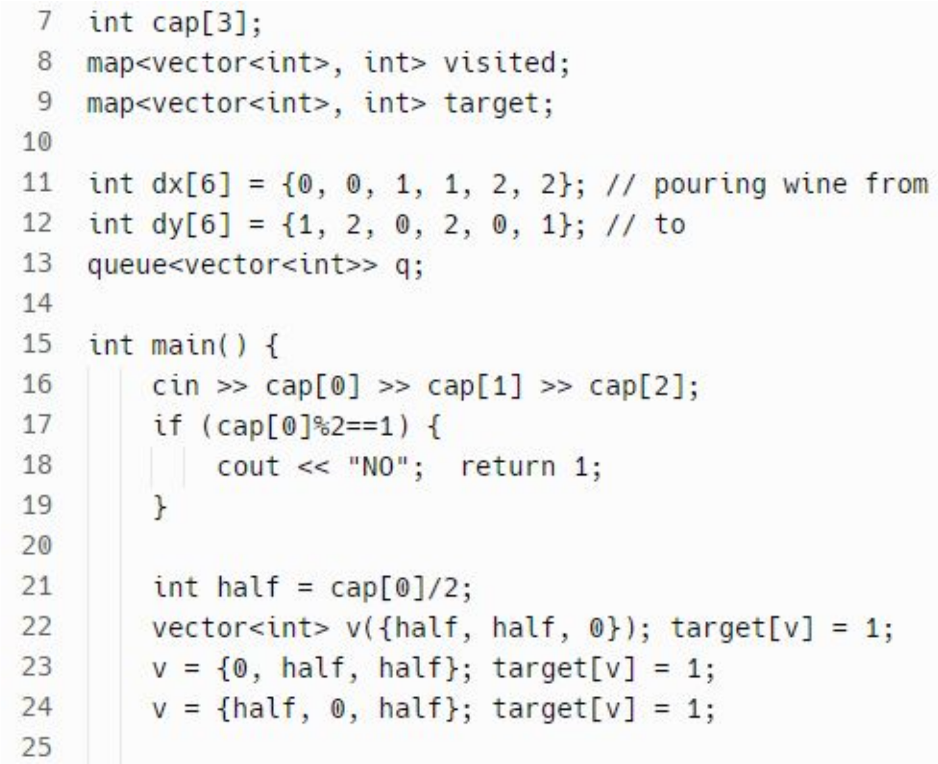


```cpp
bool is_prime(int i) {
  for(int j = 2; j <= sqrt(i); j++) {
    if(i % j == 0) {
      return false;
    }
  }
  return true;
}

int main() {
  int n;
  cin >> n;
  stack<int> stk;

  for(int i = 2; i <= n; i++) {
    if(is_prime(i)) {
      stk.push(i);
      }
  }

```

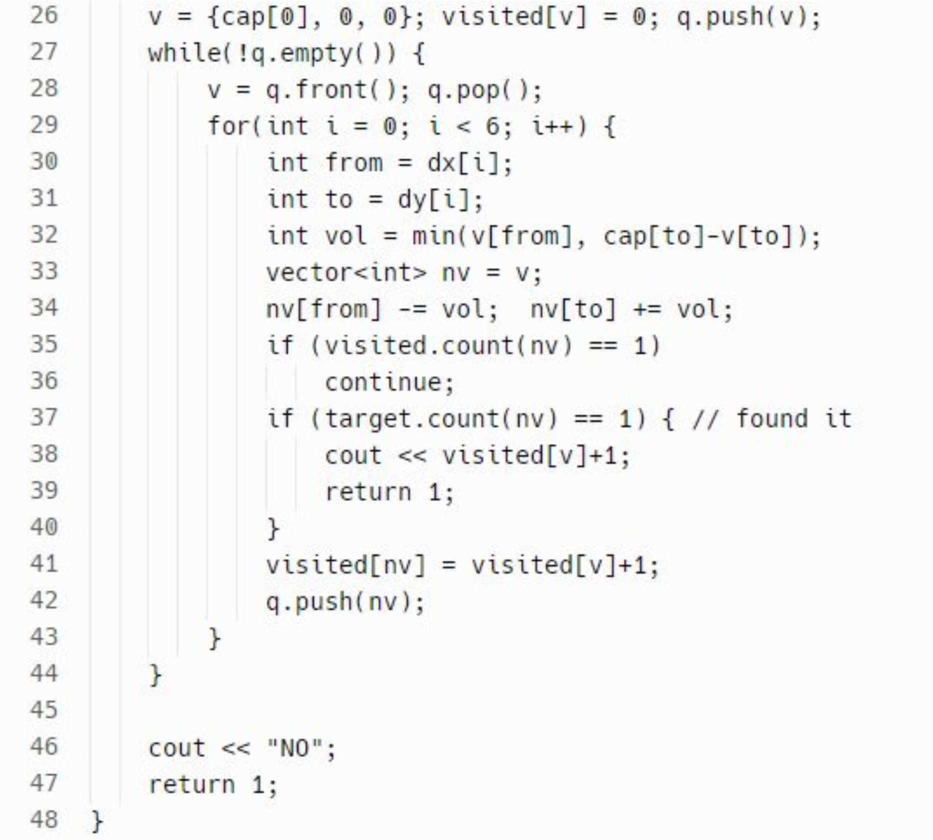
Homework - Pouring Wine - P1032



Homework - Pouring Wine - P1032



```cpp
int cap[3];
map<vector<int>, int> visited;
map<vector<int>, int> target;

int dx[6] = {0, 0, 1, 1, 2, 2}; // pouring wine from
int dy[6] = {1, 2, 0, 2, 0, 1}; // to
queue<vector<int>> q;

int main() {
    cin >> cap[0] >> cap[1] >> cap[2];
    if (cap[0]%2==1) {
        cout << "NO";  return 1;
    }

    int half = cap[0]/2;
    vector<int> v({half, half, 0}); target[v] = 1;
    v = {0, half, half}; target[v] = 1;
    v = {half, 0, half}; target[v] = 1;

```



```cpp
    v = {cap[0], 0, 0}; visited[v] = 0; q.push(v);
    while(!q.empty()) {
        v = q.front(); q.pop();
        for(int i = 0; i < 6; i++) {
            int from = dx[i];
            int to = dy[i];
            int vol = min(v[from], cap[to]-v[to]);
            vector<int> nv = v;
            nv[from] -= vol;  nv[to] += vol;
            if (visited.count(nv) == 1)
                continue;
            if (target.count(nv) == 1) { // found it
                cout << visited[v]+1;
                return 1;
            }
            visited[nv] = visited[v]+1;
            q.push(nv);
        }
    }

    cout << "NO";
    return 1;
}
```

Chess Eight (P8122)

State: the current state of the chessboard, represented by a string of 9 digits 0-8

Transitions: swap 0 with the neighboring digit

Visited: If the state has been visited

Steps: an int to track the steps moved to the state

Position of 0: track the position of 0 of the state to avoid searching it

Can move? Check if out of bound

```
typedef string state
unordered_map<state, step> visited;// or map
struct node {
  state s; // current string
  int pos; // 0 position
};

int dx[] = {-1, 1, -3, 3};

for (int i=0; i<4; i++) {
      if (i == 0 && x.pos%3 == 0) continue;
      if (i == 1 && x.pos%3 == 2) continue;
      if (i == 2 && x.pos < 3) continue;
      if (i == 3 && x.pos > 5) continue;
      int nx = x.pos + dx[i];
      state ns = x.s;
      swap(ns[x.pos], ns[nx]);
      ...
}
```

Performance comparison - P8122 Eight

Classic BFS

time: 299ms, memory: 12836kb

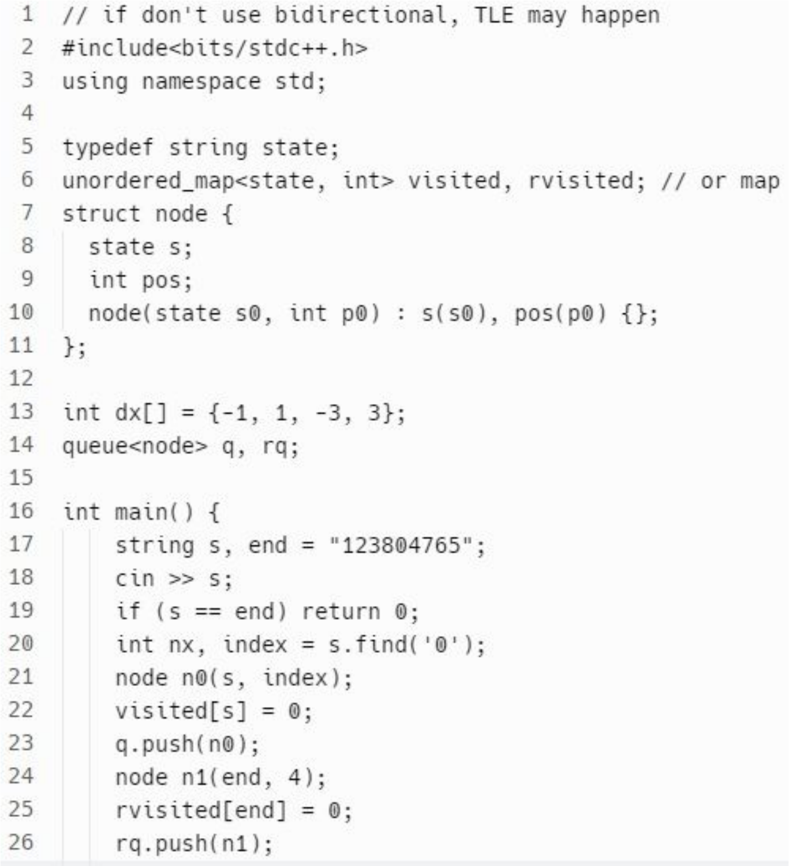
```
> test 1: time: 2ms, memory: 356kb
> test 2: time: 34ms, memory: 5648kb
> test 3: time: 12ms, memory: 2356kb
> test 4: time: 72ms, memory: 12836kb
> test 5: time: 12ms, memory: 2356kb
> test 6: time: 3ms, memory: 356kb
> test 7: time: 47ms, memory: 8536kb
> test 8: time: 85ms, memory: 10640kb
> test 9: time: 2ms, memory: 356kb
> test 10: time: 30ms, memory: 5516kb
```

Bidirectional BFS

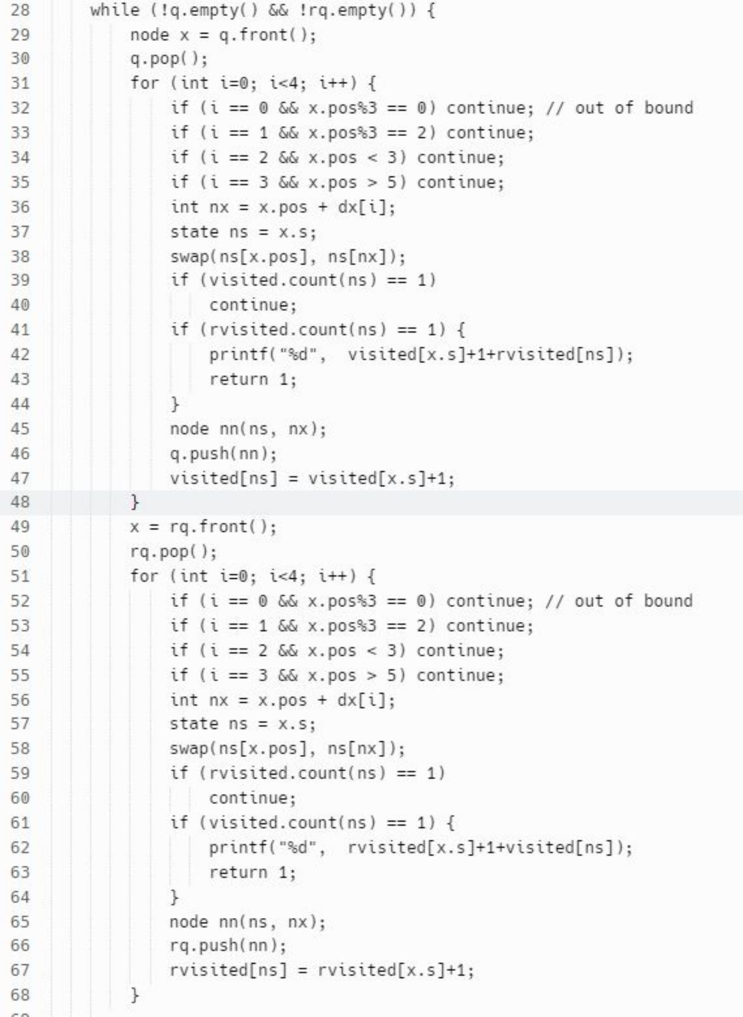
time: 29ms, memory: 1168kb

```
> test 1: time: 2ms, memory: 356kb
> test 2: time: 4ms, memory: 620kb
> test 3: time: 0ms, memory: 488kb
> test 4: time: 5ms, memory: 1168kb
> test 5: time: 3ms, memory: 488kb
> test 6: time: 3ms, memory: 356kb
> test 7: time: 1ms, memory: 752kb
> test 8: time: 5ms, memory: 1036kb
> test 9: time: 2ms, memory: 356kb
> test 10: time: 4ms, memory: 620kb
```

Chess Eight (P8122)



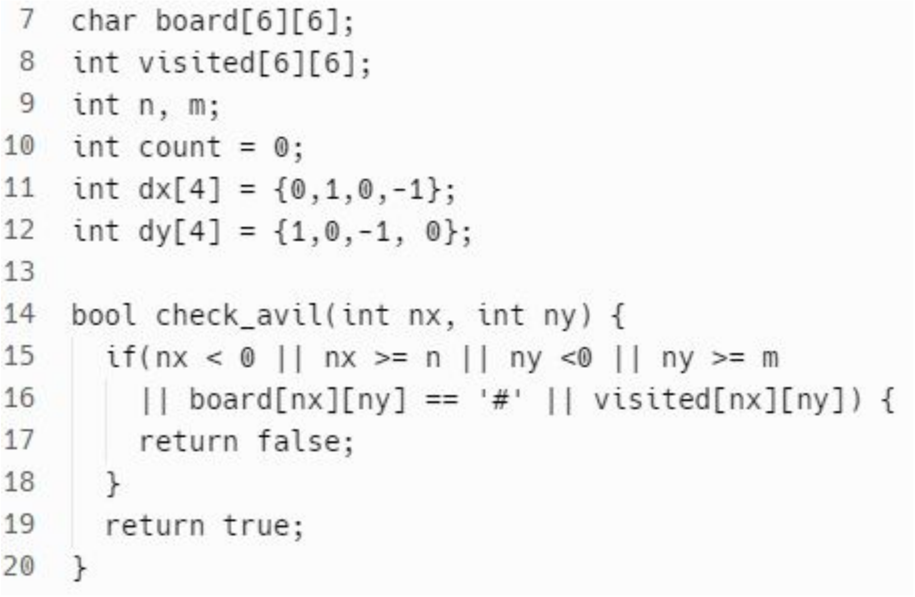
```cpp
// if don't use bidirectional, TLE may happen
#include<bits/stdc++.h>
using namespace std;

typedef string state;
unordered_map<state, int> visited, rvisited; // or map
struct node {
    state s;
    int pos;
    node(state s0, int p0) : s(s0), pos(p0) {};
};

int dx[] = {-1, 1, -3, 3};
queue<node> q, rq;

int main() {
    string s, end = "123804765";
    cin >> s;
    if (s == end) return 0;
    int nx, index = s.find('0');
    node n0(s, index);
    visited[s] = 0;
    q.push(n0);
    node n1(end, 4);
    rvisited[end] = 0;
    rq.push(n1);
```



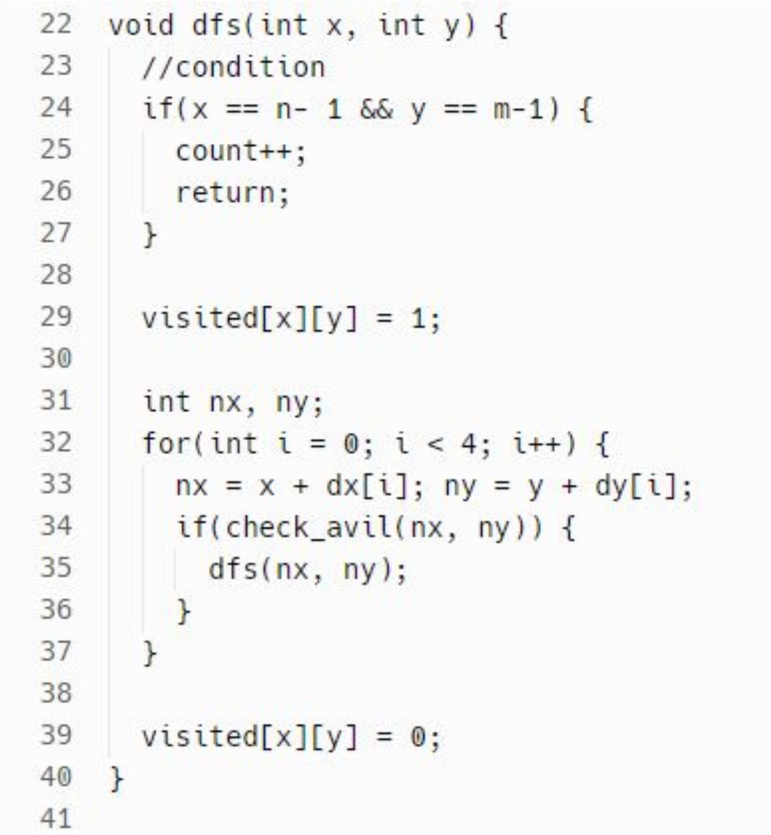
```cpp
    while (!q.empty() && !rq.empty()) {
        node x = q.front();
        q.pop();
        for (int i=0; i<4; i++) {
            if (i == 0 && x.pos%3 == 0) continue; // out of bound
            if (i == 1 && x.pos%3 == 2) continue;
            if (i == 2 && x.pos < 3) continue;
            if (i == 3 && x.pos > 5) continue;
            int nx = x.pos + dx[i];
            state ns = x.s;
            swap(ns[x.pos], ns[nx]);
            if (visited.count(ns) == 1)
                continue;
            if (rvisited.count(ns) == 1) {
                printf("%d",  visited[x.s]+1+rvisited[ns]);
                return 1;
            }
            node nn(ns, nx);
            q.push(nn);
            visited[ns] = visited[x.s]+1;
        }
        x = rq.front();
        rq.pop();
        for (int i=0; i<4; i++) {
            if (i == 0 && x.pos%3 == 0) continue; // out of bound
            if (i == 1 && x.pos%3 == 2) continue;
            if (i == 2 && x.pos < 3) continue;
            if (i == 3 && x.pos > 5) continue;
            int nx = x.pos + dx[i];
            state ns = x.s;
            swap(ns[x.pos], ns[nx]);
            if (rvisited.count(ns) == 1)
                continue;
            if (visited.count(ns) == 1) {
                printf("%d",  rvisited[x.s]+1+visited[ns]);
                return 1;
            }
            node nn(ns, nx);
            rq.push(nn);
            rvisited[ns] = rvisited[x.s]+1;
        }
```

Homework - Maze - P1032

Classic DFS



```
char board[6][6];
int visited[6][6];
int n, m;
int count = 0;
int dx[4] = {0,1,0,-1};
int dy[4] = {1,0,-1, 0};

bool check_avil(int nx, int ny) {
  if(nx < 0 || nx >= n || ny <0 || ny >= m
    || board[nx][ny] == '#' || visited[nx][ny]) {
    return false;
  }
  return true;
}
```



```
void dfs(int x, int y) {
  //condition
  if(x == n- 1 && y == m-1) {
    count++;
    return;
  }

  visited[x][y] = 1;

  int nx, ny;
  for(int i = 0; i < 4; i++) {
    nx = x + dx[i]; ny = y + dy[i];
    if(check_avil(nx, ny)) {
      dfs(nx, ny);
    }
  }

  visited[x][y] = 0;
}

```

Coding & Troubleshooting Best practices

What are your challenges?

What are your biggest challenges?

What errors do you normally encounter?

Some common errors 1/2

Error	Consequence	Comments
Variable or array not initialized	Unpredictable result	All variables and array must be initialized before usage. Global variables are initialized to 0 by default, but all local variables must be deliberately initialized.
Same global and local variable	Input/change of the local variable not effective on global variable	Let’s say n is defined as global variable and used in multiple functions, and another local variable n is defined in main and initialized.
Mixed use of i, j, m, n and so on	Crash, wrong results.	Inspect if usage of i, j, m, n, etc in wrong place.
Wrong data type	Wrong answer	Sometimes int is not large enough for calculations, then consider long long.
Wrong output format	Wrong answer	Make sure follow the exact output format

Some common errors 2/2

Error	Consequence	Comments
Out of bound - Array index <0 or >=n - queue/stack top, pop or remove without checking !empty()	Crash Segmentation fault	You must check if the index is outbound before use a[i], otherwise it will crash or sometimes unpredictable result. You also must check if an dataset is empty before access it.
Dead loop: - without increasing the variable - without popping or removing processed item - Recursion without proper stopping return	Take long time then crash	The variable controlling the loop not updated properly, e.g. - i, next value, etc in a loop - parameters passed to a recursive call The !empty() is used but the code branch doesn't pop the processed element Recursive function doesn't have proper return.

Best practices - coding

Practice	Description	Why?
Good naming convention	Give a meaningful name to each variable and function	Good variable and func names greatly help you think clearly and make code more readable. Otherwise you'll not understand your code just after a week. This also helps not misuse it.
Indents	Add indent to make the code structure clear	Indent makes the code logic structure visually clearer and easier to read and understand.

Code block and comments	Structure the code in logical sections, and add comments.	E.g.
		// initialization
		// search for the max
		// if there's still apples to eat

Functions	Organized the detailed code doing specific thing to "function"	Function tells you what a piece of code is doing, and makes it reusable, and keeps the main program flow concise and clear.

Best practices - troubleshooting

Practice	Description	Why?
Code	Follow the best coding practices	The better your code is written, the easier it is to debug.
Print out values	Print out values at key points	First make sure your code is well organized to **logic blocks**, and then it gives you a **checkpoint** after each logic block to check if everything works as expected. E.g. after read/initialization block, check if all values are expected. After add the data to dataset, e.g. map, set, vector, etc. output the dataset to check.
Test boundary	Test the least/max possible value	Try to break your code with boundary cases.
Control output for non-linear code	BFS/DFS/recursion are non-linear structure that's hard to debug	The output would be messy to just print out. You can control number of outputs using a global counter. E.g. only output first 10 times. If (counter++ <10) cout << ans << endl;
Start from a simple case	Start with a simple case that you can work manually	As needed, manually execute the simple case on paper and check against the code result.

General BFS Problems

More general BFS problems

So far we've learnt how to use BFS to solve maze problem that can be represented in simple or 2D array, but BFS can be used to solve much broader problems.

- A data structure (node) to represent start and end **state** (pair, or custom struct)
- Define **transition** rules from one node to another (which nodes are neighbors?)
- A data structure (array or map) to track if the node has been **visited**
- A data structure (array or map) to track the **distance** of the node
- Sometimes a data structure (array or map) to track the number of **paths** to the node

Pioneer

Input :

- The first line : four integers N, M, A, B --- the N*M matrix, the number of infection sources and the number of army leaders. 1<=M,N<=500, 1<=A,B<=100,000
- The next A lines: each line contains the coordinates of an infection source.
- The next B lines: each line contains the coordinates of an army leader.

Output :

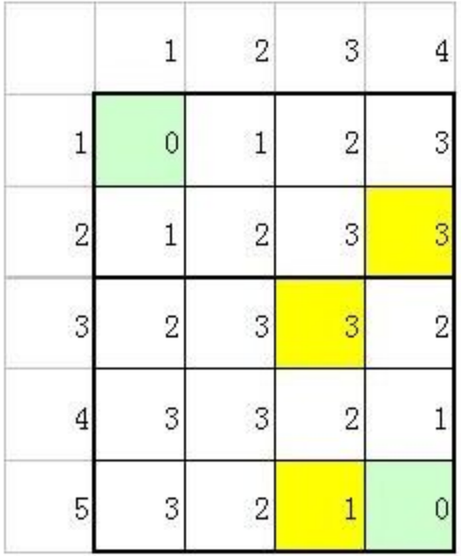
- B lines: each line contains the time for the leader to get infected.

Sample input 1:

5 4 2 3
1 1
5 4
3 3
5 3
2 4

Sample output:

3
1
3



	1	2	3	4
1	0	1	2	3
2	1	2	3	3
3	2	3	3	2
4	3	3	2	1
5	3	2	1	0

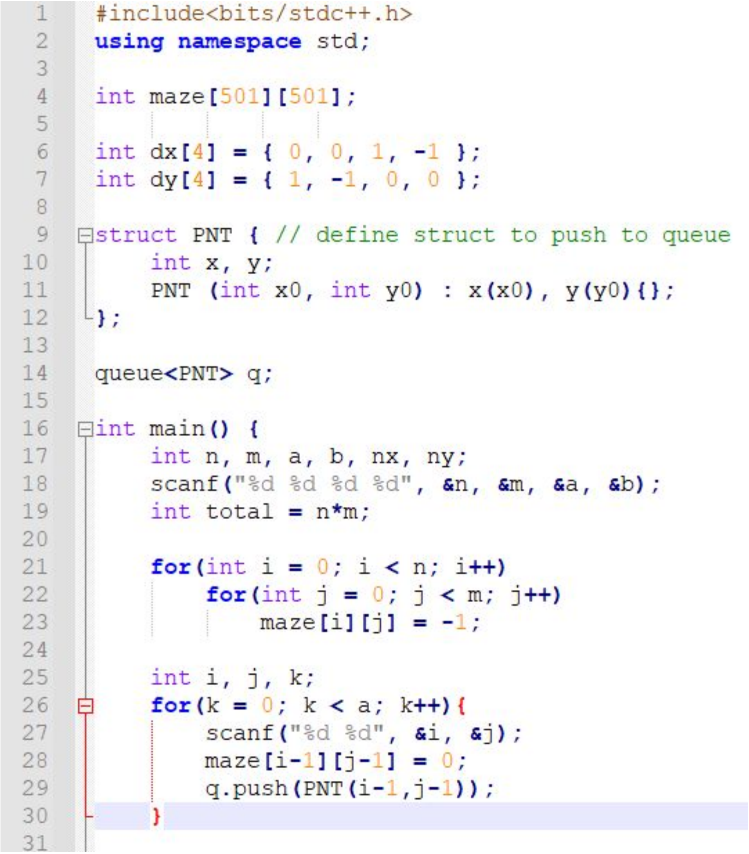
Pioneer - List key factors

Representation	Variables	Code
State	int maze[501][501] struct PNT { int x, y;}	

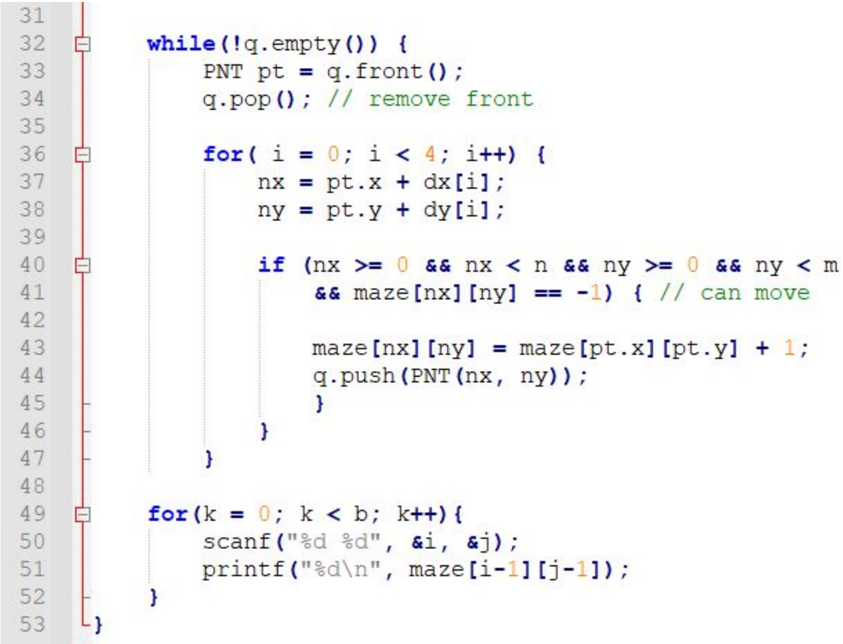
Transition	Up, down, left, right	int dx[4] = { 0, 0, 1, -1 };
		int dy[4] = { 1, -1, 0, 0 };

Visited tracking	maze[i][j] == -1 ?	if (nx >= 0 && nx < n && ny >= 0 && ny < m && maze[nx][ny] == -1)
Distance	maze[][]	maze[nx][ny] = maze[pt.x][pt.y] + 1;

Pioneer - Put them together



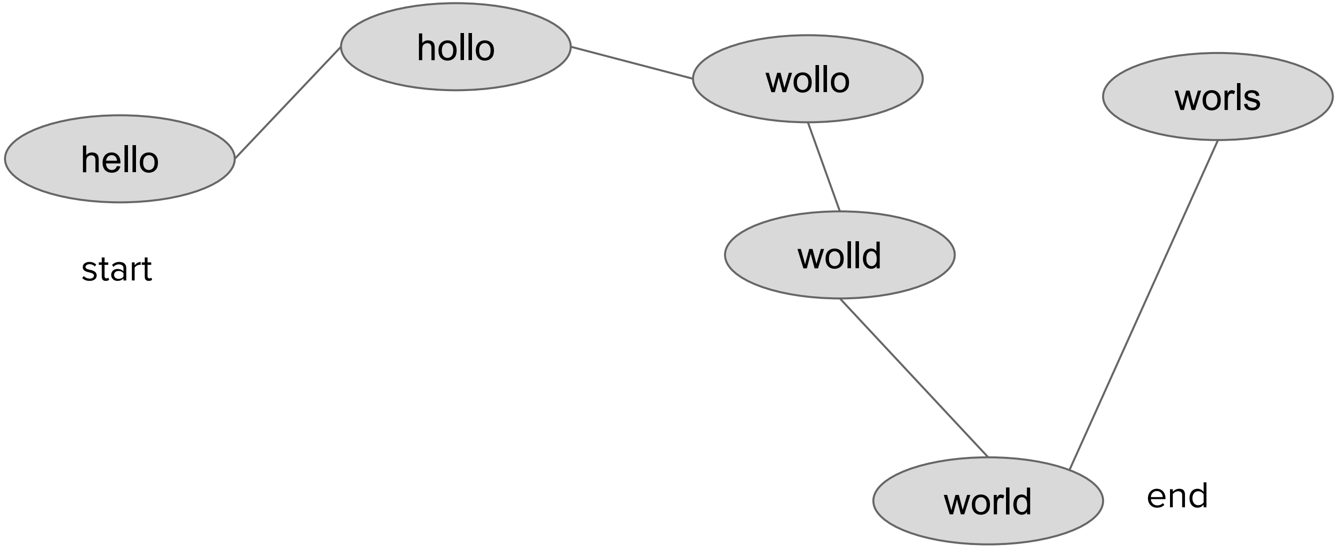
```
#include<bits/stdc++.h>
using namespace std;

int maze[501][501];

int dx[4] = { 0, 0, 1, -1 };
int dy[4] = { 1, -1, 0, 0 };

struct PNT { // define struct to push to queue
    int x, y;
    PNT (int x0, int y0) : x(x0), y(y0){};
};

queue<PNT> q;

int main() {
    int n, m, a, b, nx, ny;
    scanf("%d %d %d %d", &n, &m, &a, &b);
    int total = n*m;

    for(int i = 0; i < n; i++)
        for(int j = 0; j < m; j++)
            maze[i][j] = -1;

    int i, j, k;
    for(k = 0; k < a; k++){
        scanf("%d %d", &i, &j);
        maze[i-1][j-1] = 0;
        q.push(PNT(i-1,j-1));
    }

```



```

    while(!q.empty()) {
        PNT pt = q.front();
        q.pop(); // remove front

        for( i = 0; i < 4; i++) {
            nx = pt.x + dx[i];
            ny = pt.y + dy[i];

            if (nx >= 0 && nx < n && ny >= 0 && ny < m
                && maze[nx][ny] == -1) { // can move

                maze[nx][ny] = maze[pt.x][pt.y] + 1;
                q.push(PNT(nx, ny));
            }
        }
    }

    for(k = 0; k < b; k++){
        scanf("%d %d", &i, &j);
        printf("%d\n", maze[i-1][j-1]);
    }
}
```

Word Ladder

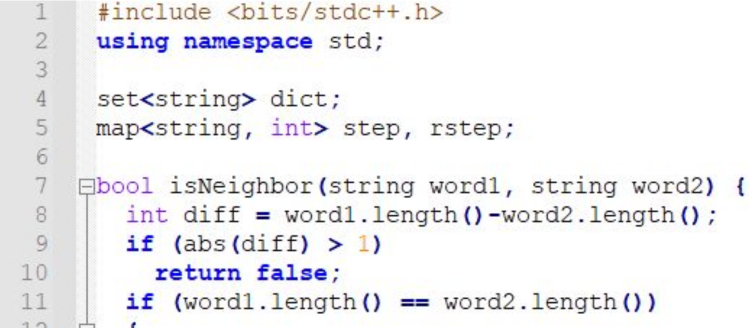
Problem transformed into - find the shortest path from start point to the destination point.



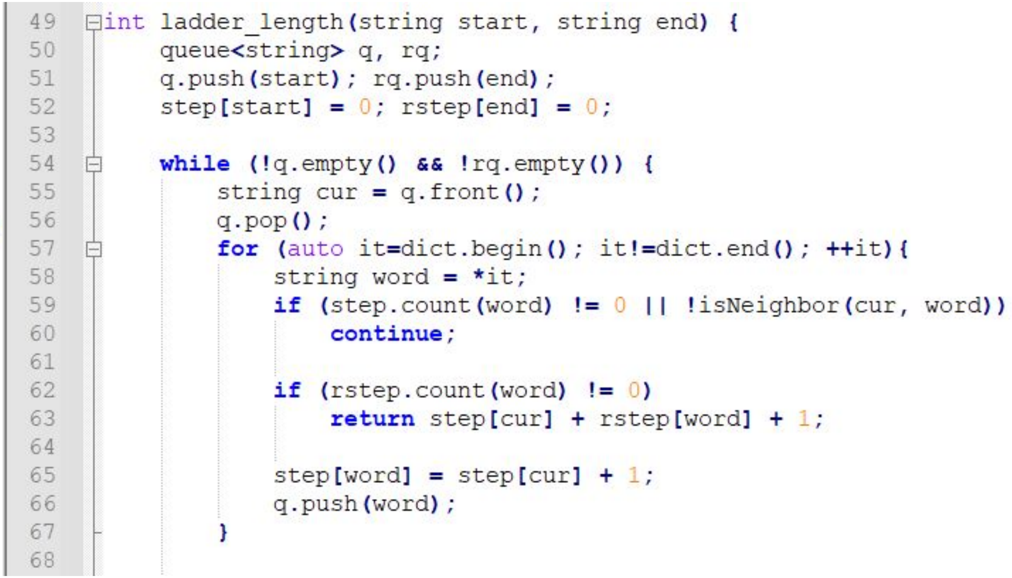
World Ladder - List key factors

Representation	**Variables**	**Code**
State	set<string> dict string word	
Transition	isNeighbor()	Go through words in dict, check if it is a neighbor of current one
Visited tracking	map<string, int> step	step.count(word) != 0
Distance	map<string, int> step	step[word] = step[cur] + 1;

World Ladder - Put them together



```
#include <bits/stdc++.h>
using namespace std;

set<string> dict;
map<string, int> step, rstep;

bool isNeighbor(string word1, string word2) {
 int diff = word1.length()-word2.length();
 if (abs(diff) > 1)
  return false;
 if (word1.length() == word2.length())
```



```
int ladder_length(string start, string end) {
  queue<string> q, rq;
  q.push(start); rq.push(end);
  step[start] = 0; rstep[end] = 0;

  while (!q.empty() && !rq.empty()) {
    string cur = q.front();
    q.pop();
    for (auto it=dict.begin(); it!=dict.end(); ++it){
      string word = *it;
      if (step.count(word) != 0 || !isNeighbor(cur, word))
        continue;

      if (rstep.count(word) != 0)
        return step[cur] + rstep[word] + 1;

      step[word] = step[cur] + 1;
      q.push(word);
    }

```

More general BFS problems

Let’s revisit the key factors

- A data structure (node) to represent start and end **state** (pair, or custom struct)
- Define **transition** rules from one node to another (which nodes are neighbors?)
- A data structure (array or map) to track if the node has been **visited**
- A data structure (array or map) to track the **distance** of the node
- A data structure (array or map) to track the number of **paths** to the node

How did we implement these in our solutions?

Pour Wine

There are three cups. We know their volumes, but there are no scales on them. The first cup is full of wine, and the others are empty. Determine whether it is possible to pour wine between them so that wine is equally divided into two parts in two cups. If it is possible, find the minimum number of steps.

Input:	Output:
8 5 3	7

The only possible end state: 4 4 0

Pour Wine

	Pioneer	Word Ladder	Pour Wine
State	int maze[501][501] struct PNT { int x, y;}	set<string> dict	vector<int> wine_volumes;
Transition	Up, down, left, right	isNeighbor()	
Visited tracking	maze[i][j] == -1 ?	map<string, int> step;	
Count	maze[][]	map<string, int> step;	

Pour Wine

	Pioneer	Word Ladder	Pour Wine

State	int maze[501][501]
	struct PNT { int x, y;}
		set<string> dict	int cup_volumes[3];
			map<vector<int>, int> target_wine_volumes;
			vector<int> wine_volumes;

Transition	Up, down, left, right	isNeighbor()	int from_cups[6] = {0, 0, 1, 1, 2, 2};
			int to_cups[6] = {1, 2, 0, 2, 0, 1};
			Either from_cup is empty, or to_cup is full.

Visited tracking	maze[i][j] == -1 ?	map<string, int> step;	map<vector<int>, int> visited_wine_volumes; Do existence check
Count	maze[][]	map<string, int> step;	map<vector<int>, int> visited_wine_volumes; Check integer value.

Chess Eight

There are 8 pieces of chess on a 3x3 chessboard, each marked with a number 1 to 8. There's one space on the chessboard marked as 0. The chess around the space can be moved to the space.

Given an initial layout (initial state) and end state (to simplify, fix it to 123804765), find the least number of the moves required to change from the initial state to the end state.

The state 123804765 means the following layout on the chessboard

```
     123
     804
     765
```

Example input: 283104765:

	Step 1	Step 2	Step 3	Step 4
283	203	023	123	123
104	184	184	084	804
765	765	765	765	765

Chess Eight

	Pioneer	Word Ladder	Chess Eight
State	int maze[501][501] struct PNT { int x, y;}	set<string> dict	string: chessboard
Transition	Up, down, left, right	isNeighbor()	
Visited tracking	maze[i][j] == -1 ?	map<string, int> step;	
Count	maze[][]	map<string, int> step;	

Chess Eight

	Pioneer	Word Ladder	Chess Eight

State	int maze[501][501]	set<string> dict	struct Chessboard {
			  string chessboard;
			  int zero_pos;
			};

Transition	Up, down, left, right	isNeighbor()	int dx[] = {-1, 1, -3, 3};
Visited tracking	maze[i][j] == -1 ?	map<string, bool> visited;	map<string, int> forward_visited, backward_visited; Do existence check.
Count	maze[][]	map<string, int> step;	map<string, int> forward_visited, backward_visited; Check integer value.

More Problems (HW)

Multiple of N (P1339)

Description:
Write a program to find a minimum positive multiple of **N** for a given natural number **N** (1 ≤ N ≤ 4999) and **M** different decimal numbers X_1, X_2, ..., X_M (at least one) so that there are no numbers other than X_1, X_2, ..., X_M in this multiple.

Input format:
1st line: **N**
2nd line: **M**
Followed by **M** lines: the **M digits** that can be in the multiple.

Output :
Output this multiple, if there is no solution output 0.

Data range:
1≤N≤4999
Restrictions:
The answer will not exceed 500 digits in all test data.

- Use the Key Factors strategy
- How to construct numbers using M digits, and test if the constructed number is divisible by N?
- Note the constructed number may be **very large**

https://xjoi.net/contest/4776

Sample input:

```
4999
4
7
6
9
0
```

Sample output:

```
60007996
```

Multiple of N

How to construct numbers using M numbers, and test if the constructed number is divisible by N?

- List key factors
- Find transition way
- Sort M digits
- Get starting states
- Grow the constructed number by appending a digit
- When to stop searching?

How to test divisibility of large numbers?

- Possible way, testing remainder

```
struct node {
  string number;
  int remainder;
};
```

Sample input:

```
4999
4
7
6
9
0
```

Sample output:

```
60007996
```

Graph Basics

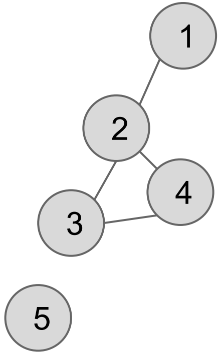
Graphs

- A data structure that consists of a set of nodes (vertices) and a set of edges that relate the nodes to each other
- The set of edges describes relationships among the vertices
- A graph G is defined as follows:

$G=(V,E)$

$V(G)$: a finite, nonempty set of vertices

$E(G)$: a set of edges (pairs of vertices)



Example

$G=(V,E)$

$V=\{1,2,3,4,5\}$

$E=\{(1,2), (2,3), (2,4), (3,4)\}$

Directed vs. Undirected Graphs

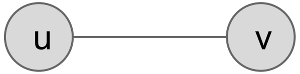
Undirected edge has no orientation (no arrow head)

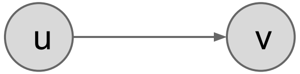
Directed edge has an orientation (has an arrow head)

Undirected graph – all edges are undirected

Directed graph – all edges are directed

Any example in our life?



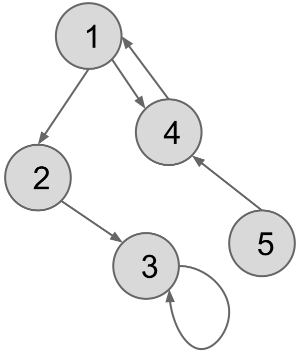


Examples of Graph

Directed Graph $G=(V,E)$

$V=\{1,2,3,4,5\}$

$E=\{(1,2), (2,3), (1,4), (4,1), (3,3), (5,4)\}$



Undirected Graph $G=(V,E)$

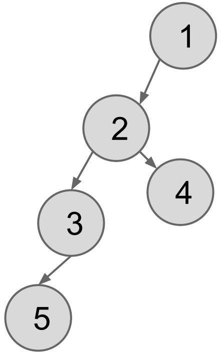
$V=\{1,2,3,4,5\}$

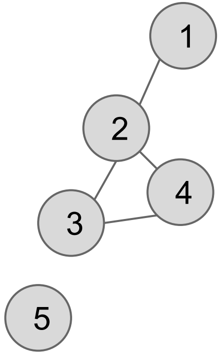
$E=\{(1,2), (2,3), (2,4), (3,4)\}$

Directed Graph $G=(V,E)$

$V=\{1,2,3,4,5\}$

$E=\{(1,2), (2,3), (2,4), (3,5)\}$





Terminologies

For directed graph

- Edge (i, j) is **incident to** vertex j and **incident from** vertex i, meanwhile vertex i is **adjacent to** vertex j, and vertex j is **adjacent from** vertex i.
- **In-degree** of vertex i is the number of edges incident to i
- **Out-degree** of vertex i is the number of edges incident from i

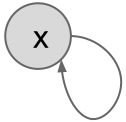
For undirected graph

- Edge (i, j) is **incident on** vertex j and vertex i, meanwhile vertex i is **adjacent to** vertex j, and vertex j is **adjacent to** vertex i
- The **degree** of vertex i is the number of edges incident on vertex i.

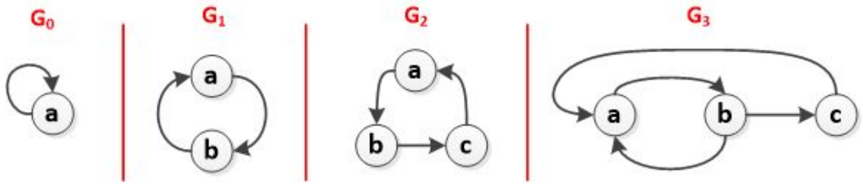
Path: a sequence of vertices that connect two nodes in a graph.

A **simple path** is a path in which all vertices, except possibly in the first and last, are different.

Terminologies



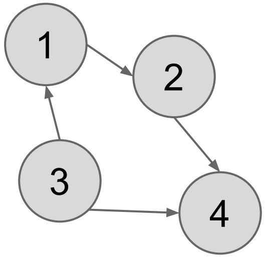
Loop — edges that connect a vertex to itself.
A **cycle** is a simple path with the same start and end vertex.



Multiple Edges: two nodes may be connected by >1 edge (for directed graph, (x,y) (y,x) are not considered multiple edges)
Simple Graphs: have no loops and no multiple edges
Connected Graph: a graph in which it's possible to get from every vertex in the graph to every other vertex through a path.

Graph Representation

How do we represent graphs using data structure?



Directed graph:

$G=(V,E)$

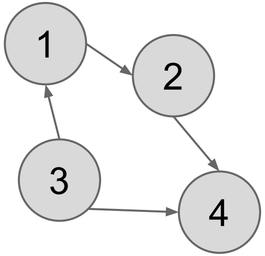
$V=\{1,2,3,4\}$

$E=\{(1,2), (2,4), (3,1), (3,4)\}$

Graph Representation

There are typically two computer representations of graphs:

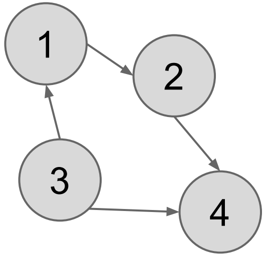
- Adjacency matrix representation
- Adjacency lists representation



Adjacency Matrix

- A square grid of boolean values
- If the graph contains N vertices, then the grid contains N rows and N columns
- For two vertices numbered i and j, the element at row i and column j is true if there is an edge from i to j, otherwise false

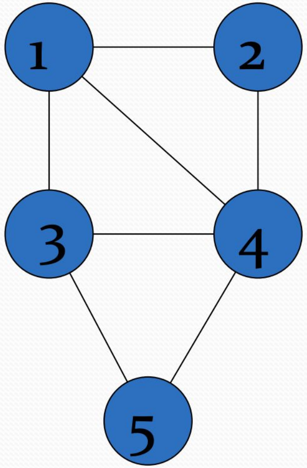
	Vert 1	Vert 2	Vert 3	Vert 4
Vert 1	0	1	0	0
Vert 2	0	0	0	1
Vert 3	1	0	0	1
Vert 4	0	0	0	0

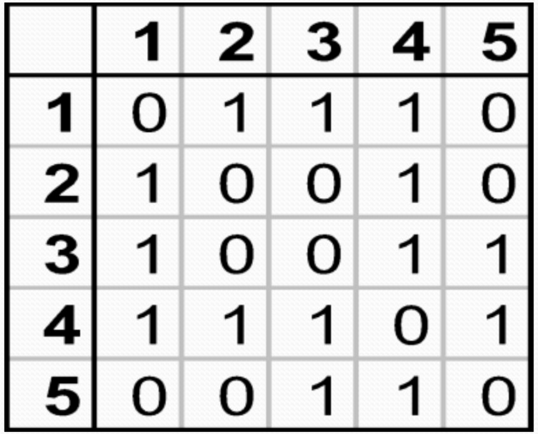


Depth First Search on adjacency matrix

```
bool is_visited[N+1]; // Indicates a vertex is visited or not during DFS, starting with all false.
bool matrix[N+1][N+1]; // Adjacency matrix.
```

```
void dfs(int current) {
 is_visited[current] =  true;
 for (int i = 1; i <= N; i ++) {
   if (matrix[current][i] &&
       !is_visited[i]) {
     dfs(i);
   }
 }
}
```





	1	2	3	4	5
1	0	1	1	1	0
2	1	0	0	1	0
3	1	0	0	1	1
4	1	1	1	0	1
5	0	0	1	1	0

Space Complexity O(n^2)

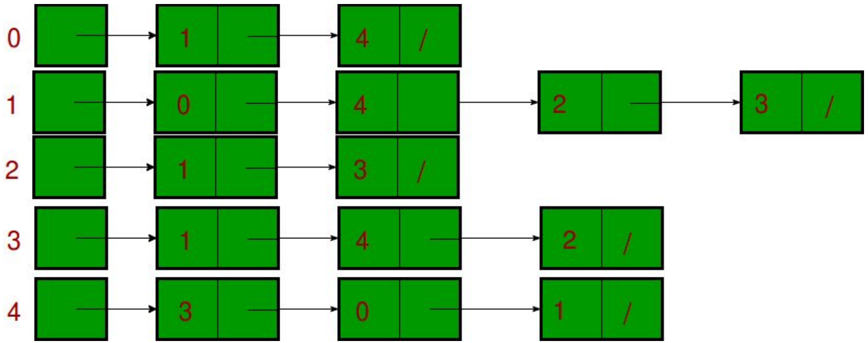
Adjacency Lists Representation

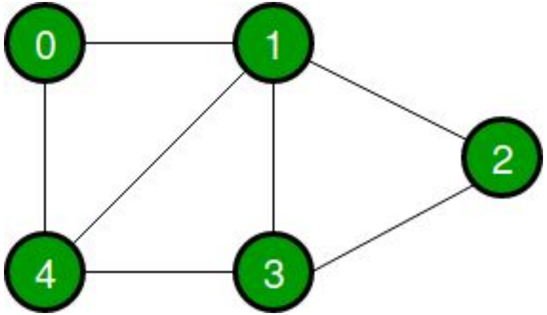
A graph of n nodes is represented by a one dimensional array L of linked lists, where

- L[i] is the linked list containing all the nodes adjacent from node i.
- The nodes in the list L[i] are in no particular order

Example:

- L[0] = {1,4}
- L[1] = {0,4,2,3}
- L[2] = {1,3}
- L[3] = {1,4,2}
- L[4] = {3,0,1}

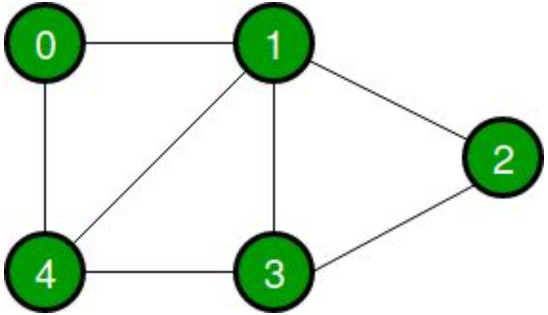




Common implementations: use vector<int> L[]; or list<int> L[];.

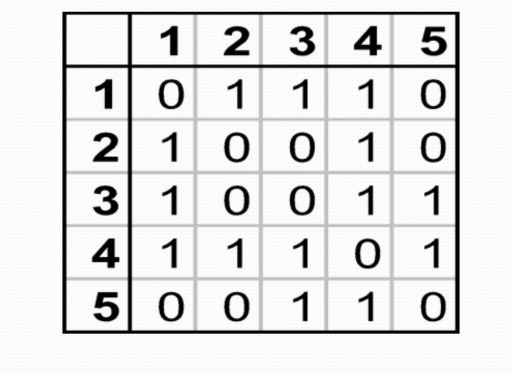
Adjacency Lists — Adding an Edge & DFS

```
const int N = 1000010;
vector<int> adj[N]; // Adjacency lists.
bool is_visited[N]; // Indicates a vertex is visited or not during DFS.

void add_edge(int x, int y) {
 adj[x].push_back(y);
 adj[y].push_back(x); // For undirected graph, adds both directions.
}

void dfs(int current) {
 is_visited[current] = true;
 for(auto x : adj[current]) {
   if (!is_visited[x]) {
     dfs(x);
   }
 }
}
```

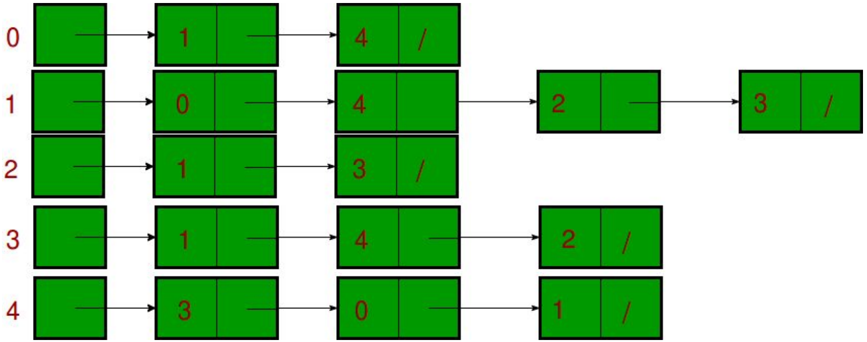


Graph Representation

What are pros/cons for the two representations?



	1	2	3	4	5
1	0	1	1	1	0
2	1	0	0	1	0
3	1	0	0	1	1
4	1	1	1	0	1
5	0	0	1	1	0



Graph USACO References

https://usaco.guide/bronze/intro-graphs?lang=cpp

Livestock Lineup - P15360

INPUT FORMAT:

The first line of input contains N. The next N lines each contain a sentence describing a constraint in the form "X must be milked beside Y", where X and Y are names of some of Farmer John's cows (the eight possible names are listed above).

OUTPUT FORMAT:

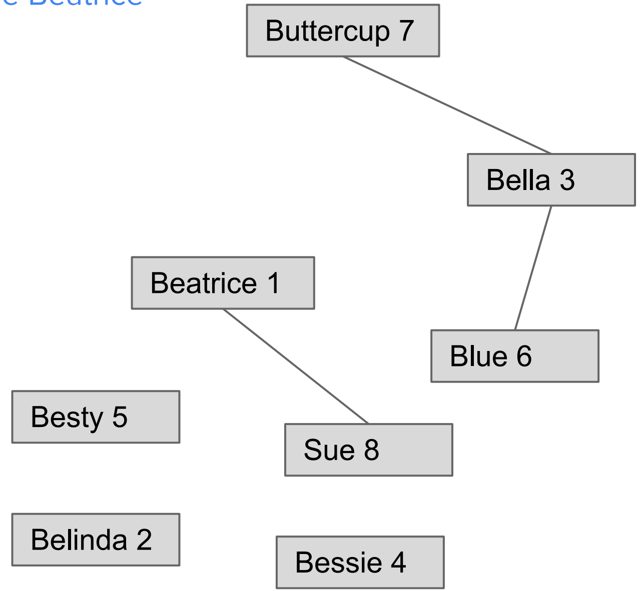
Please output, using 8 lines, an ordering of cows, one cow per line, satisfying all constraints. If multiple orderings work, output the one that is alphabetically earliest.

SAMPLE INPUT:

```
3
Buttercup must be milked beside Bella
Blue must be milked beside Bella
Sue must be milked beside Beatrice
```

SAMPLE OUTPUT:

```
Beatrice
Sue
Belinda
Bessie
Betsy
Blue
Bella
Buttercup
```



Buttercup 7

Bella 3

Beatrice 1

Blue 6

Besty 5

Sue 8

Belinda 2

Bessie 4

Livestock Lineup - P15360

Some questions to answer:

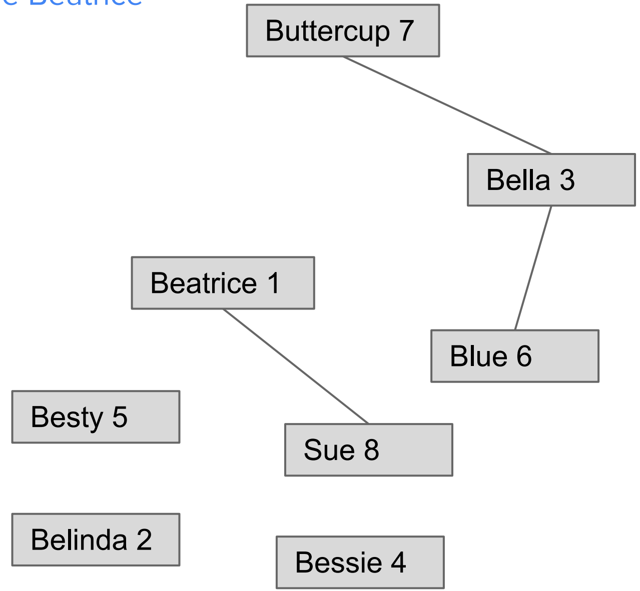
- How do we use graph concept?
- Directed graph or undirected graph?
- How to build data structure?
- Some cows may not be lead, e.g. Bella
- How do we choose lead?
- How do we expand to others?

SAMPLE INPUT:

```
3
Buttercup must be milked beside Bella
Blue must be milked beside Bella
Sue must be milked beside Beatrice
```

SAMPLE OUTPUT:

```
Beatrice
Sue
Belinda
Bessie
Betsy
Blue
Bella
Buttercup
```


Buttercup 7
Bella 3
Beatrice 1
Blue 6
Besty 5
Sue 8
Belinda 2
Bessie 4