Teamscode Contest

- Aug 06, 8 am ~ 3:30 pm, Coding time 8:30 ~ 11:30 pm
- Three levels, Novice, Intermediate, Advanced
- 202 B will be in Novice level
- Great opportunity for teamwork, practice and have fun
- Make up your team and register at https://docs.google.com/spreadsheets/d/1R-HxMLzKPWUHvmqDJVM_qCsA39hmiz-QppQhhZoaAuQ/edit#gid=1062293368
- Consider to make team with your study buddies
- If you don't have a team, register and X-Camp can help to form a team

- Early bird deadline for fall registration is this Sunday.

Cow cross road (P4065)

SAMPLE INPUT:

```
8
3 1
3 0
6 0
2 1
4 1
3 0
4 0
3 1
```

SAMPLE OUTPUT:

```
3
```

```
4  int main() {
5      int n, cow, side, cross = 0;
6      cin >> n;
7      map<int, int> mymap;
8      for(int i = 0; i < n; i++) {
9          cin >> cow >> side;
10         if (mymap.count(cow) == 0)
11             mymap[cow] = side;
12         else if (mymap[cow] != side) {
13             mymap[cow] = side;
14             cross++;
15         }
16     }
17     cout << cross;
18 }
```

Pioneer Problem P8119

Multiple sources

Sample input 1:

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

Sample output 1:

```
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 Problem P8119

Time Limit Exceeded? *scanf/printf*

```
17  int main() {
18      int n, m, a, b, nx, ny;
19      scanf("%d %d %d %d", &n, &m, &a, &b);
20      int total = n*m;
21
22      for(int i = 0; i < n; i++)
23          for(int j = 0; j < m; j++)
24              maze[i][j] = -1;
25
26      int i, j, k;
27      for(k = 0; k < a; k++){ // add all sources
28          scanf("%d %d", &i, &j);
29          maze[i-1][j-1] = 0;
30          q.push(PNT(i-1,j-1));
31      }
```

```
33      while(!q.empty()) {
34          PNT pt = q.front();
35          q.pop(); // remove front
36
37          for( i = 0; i < 4; i++) {
38              nx = pt.x + dx[i];
39              ny = pt.y + dy[i];
40
41              if (nx >= 0 && nx < n && ny >= 0 && ny < m
42                  && maze[nx][ny] == -1) { // can move
43
44                  maze[nx][ny] = maze[pt.x][pt.y] + 1;
45                  q.push(PNT(nx, ny));
46                  }
47              }
48          }
49
50      for(k = 0; k < b; k++){
51          scanf("%d %d", &i, &j);
52          printf("%d\n", maze[i-1][j-1]);
53      }
54  }
```

Bi-directional BFS for word ladder

```
queue<node> q, rq;
q.push(start);     rq.push(end);
while (!q.empty() && !rq.empty()) {
        head = q.front();       q.pop();
        Enqueue all words from dictionary that head can get to with one step (one modification)
        Termination check (whether the word has appeared in rq)
        Infinity circle check ← add to q

        head = rq.front();              rq.pop();
        Enqueue all words words from dictionary that head can get to with one step (one modification)
        Termination check (whether the word has appeared in q)
        Infinity circle check ← add to rq
    }
```

Word Ladder - check two words

- For word in dictionary, check if it is neighbor of current word
- Are **hello** and **hollo** neighbors? How about **helloo**? **hellooo**? **hell**? **hel**?
- check lengths, if difference > 1, false
- if difference == 1
  - remove element from longer string to see if they match
- if same length
  - check how many elements are different
  - diff == 1 ➔ yes; otherwise ➔ no

```
4  set<string> dict;
5  map<string, int> step, rstep;
6
```

Word Ladder (P12608)

```cpp
// bidirectional BFS
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);
    }

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

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

      rstep[word] = rstep[cur] + 1;
      rq.push(word);
    }
  }
  return -1;
}
```

```cpp
bool isNeighbor(string word1, string word2) {
  int diff = word1.length()-word2.length();
  if (abs(diff) > 1) // length diff more than 1
    return false;
  if (word1.length() == word2.length()) {
    int count = 0;
    for (int i = 0; i < word1.length(); i++) {
      if (word1[i] != word2[i])  {
        count++;  // number if diff
        if (count > 1)
          return false;
      }
    }
    return true;
  }
  else if (word1.length() == word2.length()+1) {
    for (int i = 0; i < word1.length(); i++) {
      string tmp = word1;
      tmp.erase(i, 1);
      if (tmp == word2)
        return true;
    }
    return false;
  }
  else {
    for (int i = 0; i < word2.length(); i++) {
      string tmp = word2;
      tmp.erase(i, 1);
      if (tmp == word1)
        return true;
    }
    return false;
  }
}
```

Log (P7669)

Sample Input:

```
13
0 1
0 2
2
0 4
0 2
2
1
2
1
1
2
1
2
```

Format 1: 0 X // One container stock-in Oper
storage.

Format 2: 1 // One container stock-out. Thi
container.

Format 3: 2 // A query operation, requiring

Sample Output:

```
2
4
4
1
0
```

How to use stack to track storage? How to track the largest one

Log (P7669)

Sample Input:

```
13
0 1
0 2
2
0 4
0 2
2
1
2
1
1
2
1
2
```

Format 1: 0 X // One container stock-in Oper
storage.

Format 2: 1 // One container stock-out. Thi
container.

Format 3: 2 // A query operation, requiring

Sample Output:

```
2
4
4
1
0
```

```cpp
// here we push the current greatest weight into stack
int main() {
    stack<int> st;
    int n, x1, x2 = 0;
    scanf("%d", &n);

    while (n > 0) {
        scanf("%d", &x1);
        if (x1 == 0) {
            scanf("%d", &x2);
            if (st.empty())
                st.push(x2);
            else
                st.push(max(x2, st.top()));
        }
        else if (x1 == 1 && !st.empty())
            st.pop();
        else if (x1 == 2) {
            if (!st.empty())
                printf("%d\n", st.top());
            else
                printf("0\n");
        }
        n--;
    }
}
```

Selected Advanced Search Techniques

X-Camp

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.

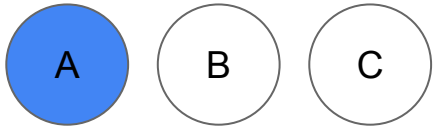
Input:	Output:
8 5 3	7

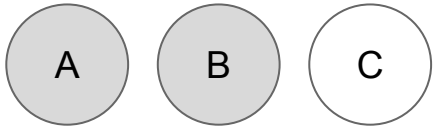
The only possible end state: 4 4 0

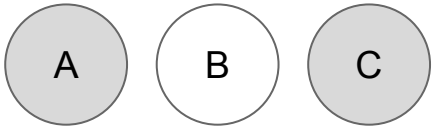
800->350->323->620->602->152->143->440

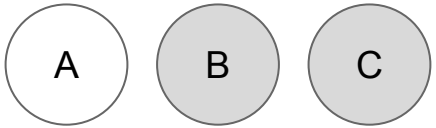
Start state and end (target) States

In the end, the wine can be in any two of three cups. There are three ways to select two cups from three cups, so there are three target states. We can initially push these states into rq.









Can we solve this problem using Bidirectional BFS?

Pruning

In some cases, when we encounter a state, we can immediately tell that this state will never lead to a solution, or the solution will not be optimal. This indicates that we can ignore the further states of this state, and we do not need to push them into queue.

Such process is called pruning.

Familiar? Did we already implement this but in another term?

Pruning for Maze Problem

If someone step out of the board, or step into a blocker cell, then we do not need to push this state into queue, because there will be no solution – **out of bound**.

If someone step into some grid which has already been visited by some other person, then we do not need to push this state into queue, because it will not be optimal – **visited**.

Pruning for Word Ladder

If we get a word that is not in the directory, then there is no solution for this state.

If we get a word that has already appeared in the same queue, then the solution will not be optimal.

BFS with Different Costs

In maze problem, the cost of walking from one cell to another is 1. In word ladder problem, the cost of changing word once is 1. It means that each step has equal cost.

What if the costs are different for different step? For example, in maze problem, the cost of walking from one cell to another depends on these two cells.

Any example in our life?

Optimal Solution

If the costs are different, then we cannot say that the first solution we find must be an optimal solution.

Rather, we can use the cost for that solution as a threshold. If we reach a state where the cost is greater than the threshold, then this state will never lead to an optimal solution.

BFS Pseudocodes with different cost

```
answer = ∞
queue<node> q;
q.push(start);
while (!q.empty()) {
        head = q.front();
        q.pop();
        If the cost of head is greater than answer, then skip current loop and continue
        If head is one of the target states, then answer = min(answer, head.cost)
        Otherwise, enqueue states that can be reached from head in one step
}
```

BFS with different cost - Practice

// if 0, cost to move is 1, otherwise, cost is 1+number

// find the min cost from maze[0][0] to maze[4][4].

```
int maze[5][5] = {{0, 0, 0, 0,6 },
                  {0,10,10,10,0 },
                  {0,10, 0, 0,0 },
                  {0,10, 0,10,0 },
                  {0, 0, 0,10,0 } };
```

https://onlinegdb.com/SXS9qAFuQ In real world: find a driving route without traffic

BFS with different cost - Practice

// sample code, https://onlinegdb.com/xhF7emVx4

// find the min cost from maze[0][0] to maze[4][4]

```
int maze[5][5] = {{0, 0, 0, 0,6 },
                  {0,10,10,10,0 },
                  {0,10, 0, 0,0 },
                  {0,10, 0,10,0 },
                  {0, 0, 0,10,0 } };
```

Chess Eight (P8122)

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

https://www.bilibili.com/video/BV1kW411t7S5?spm_id_from=333.337.search-card.all.click

Sample input:
283104765

Sample output:
4

Note:
283104765 means
283
104
765

the 3x3 chessboard

Chess Eight (P8122)

What's given?

- A initial state of a 3x3 chessboard with 8 pieces marked from 1 to 8, and a space marked 0.
- An end state to transition to
- The rules to move the pieces

What's asked?

- The least number of the moves to achieve the end state

What's your strategy?

- What's the state and how to represent?
- What are the transitions from each state?
- DFS or BFS?

Sample input:

283104765

Sample output:

4

Note:

283104765 means

283

104

765

nn the 3x3 chessboard

Chess Eight (P8122)

Considering the coding?

How do we represent a state?

Special handling: know where ‘0’ is

Is this state visited?

What is step to a state?

How to find transition of current state?

How to move to next state?

dx[] = {1, -1, 3, -3}?

Sample input:
283104765

Sample output:
4

Note:
283104765 means
283
104
765
nn the 3x3 chessboard

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