Introduction to Competitive Programming

MCPC-2020 Winter Training

Shizhe Zhao (eggeek)

About the training

- There will be 2 - 3 workshops in winter
- To cover: problem solving, C++ and data structures
- Training contests will be on: `https://vjudge.net/group/monashicpc`

Note that, this training **is not**:

- for your coursework
- for tech-interview and software development
- although it helps things above, a lot.

It **is**

- developing your problem-solving skill
- filling up the gap between your coursework and practice

Basic Strategy: Brute-force

- Helps you understand the problem
- Good start-point before implementing a complicate algorithm
- Examples ...

Brute-force example: anticolockwise motion *(SP-ICPC 2016)*

21	20	19	18	17
22	7	6	5	16
23	8	1	4	15
24	9	2	3	14
25	10	11	12	13

Q: What's the location of $n \leq 10^{10}$?

Brute-force example: anticolockwise motion *(SP-ICPC 2016)*

21	20	19	18	17
22	7	6	5	16
23	8	1	4	15
24	9	2	3	14
25	10	11	12	13

Brute-force thinking ($O(n)$):
simulate the "motion" for n steps.

Brute-force example: anticolockwise motion *(SP-ICPC 2016)*

21	20	19	18	17
22	7	6	5	16
23	8	1	4	15
24	9	2	3	14
25	10	11	12	13

Brute-force thinking ($O(n)$):
simulate the "motion" for n steps.

Popping up sub-questions:

- start-up direction?
- #num of steps?
- when direction change?

Brute-force example: anticolockwise motion *(SP-ICPC 2016)*

21	20	19	18	17
22	7	6	5	16
23	8	1	4	15
24	9	2	3	14
25	10	11	12	13

Popping up sub-questions:

- start-up direction?
- #num of steps?
- when direction change?

Wait, since you know **#num of steps**, you don't need to simulate each motion ...

Brute-force example: anticolockwise motion *(SP-ICPC 2016)*

21	20	19	18	17
22	7	6	5	16
23	8	1	4	15
24	9	2	3	14
25	10	11	12	13

Popping up sub-questions:

- start-up direction?
- #num of steps?
- when direction change?

Wait, since you know **#num of steps**, you don't need to simulate each motion ...

Which will be $O(\sqrt{n})$!

Brute-force example: runtime analysis

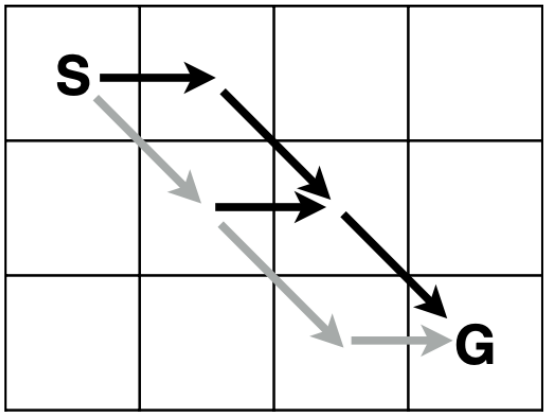
Keep in mind:
You can assume, C++ simulates 10^8 (Python: 10^6) "steps"(i.e. CPU operations) per second,

Thus, the estimated time of an $O(n)$ solution is **100s** when $n = 10^{10}$, while an $O(\sqrt{n})$ solution is **0.001s**; there are orders of magnitude difference!

Brute-force example: further thinking

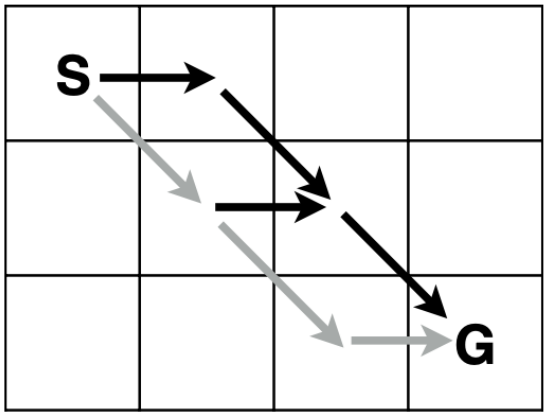
- Can we do this better than $O(\sqrt{n})$?
- What if the moving-pattern is in rectangle instead of square?
- What if cells are: *triangles*, *hexagons*?

Anti Brute-force example: Fast and Furious* *(MCPC 2020)*



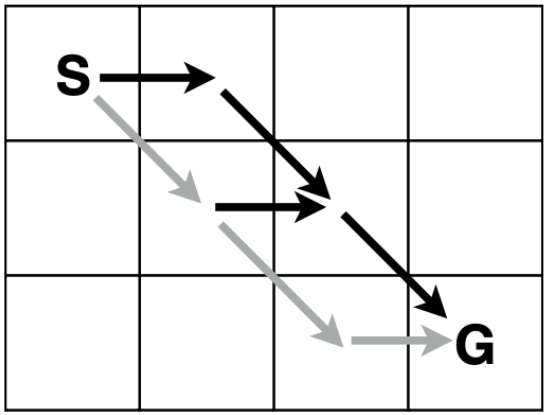
- Q: In infinite grid-map, given **m**, find a path that minimize the steps from **S** to **G**, while direction must be different at second $\mathbf{k} * \mathbf{m}$ and $\mathbf{k} * \mathbf{m} + \mathbf{1}$, for any $k > 0$.

Anti Brute-force example: Fast and Furious* *(MCPC 2020)*



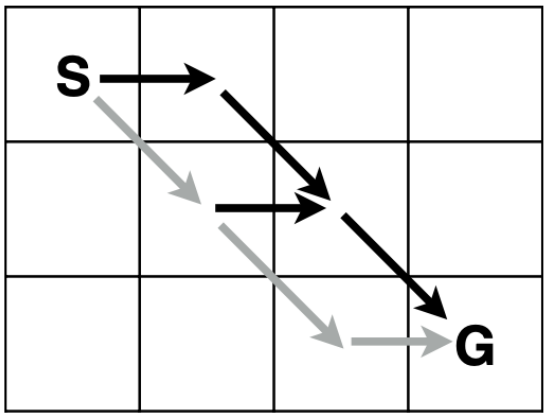
- Q: In infinite grid-map, given **m**, find a path that minimize the steps from **S** to **G**, while direction must be different at second $\mathbf{k} * \mathbf{m}$ and $\mathbf{k} * \mathbf{m} + \mathbf{1}$, for any $k > 0$.
- Brute-force thinking:
  Move on the shortest path (*on which?*), change direction (*to which?*) when necessary (*when?*).

Anti Brute-force example: Fast and Furious* *(MCPC 2020)*



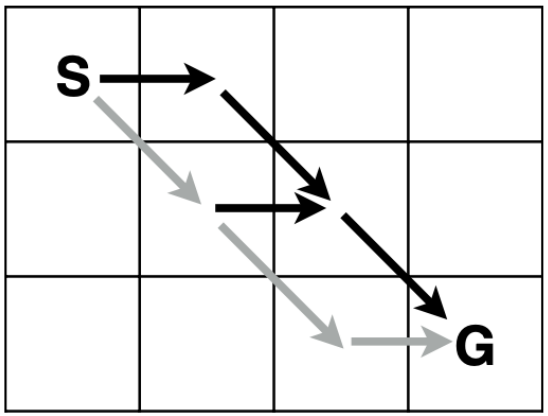
- Q: In infinite grid-map, given **m**, find a path that minimize the steps from **S** to **G**, while direction must be different at second $\mathbf{k} * \mathbf{m}$ and $\mathbf{k} * \mathbf{m} + \mathbf{1}$, for any $k > 0$.
- Brute-force thinking:
  Move on the shortest path (*on which?*), change direction (*to which?*) when necessary (*when?*).
- You will able to write a program now.
  But wait, **how to guarantee correctness?**

Anti Brute-force example: Fast and Furious* *(MCPC 2020)*



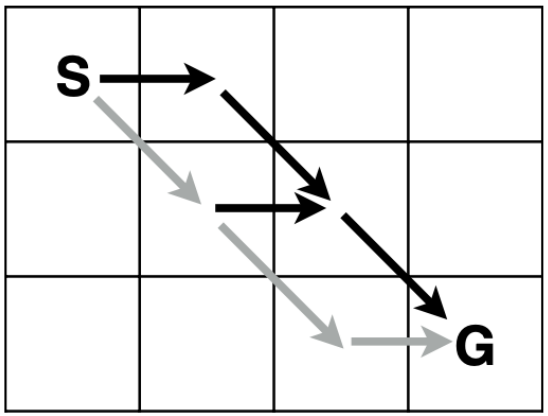
- Approach 1 (brute-force):
  This problem is actually in 3D space - *(x, y, time)*; and your brute-force simulation should based on this.
  (conceptually easier, practically harder)

Anti Brute-force example: Fast and Furious* *(MCPC 2020)*



- Approach 1 (brute-force):
  This problem is actually in 3D space - *(x, y, time)*; and your brute-force simulation should based on this.
  (conceptually easier, practically harder)
- Approach 2 (greedy):
  Minimize $|dx - dy|$ in each step.
  (conceptually harder, practically easier)

Anti Brute-force example: Fast and Furious* *(MCPC 2020)*



- Approach 1 (brute-force):
  This problem is actually in 3D space - *(x, y, time)*; and your brute-force simulation should based on this.
  (conceptually easier, practically harder)
- Approach 2 (greedy):
  Minimize $|dx - dy|$ in each step.
  (conceptually harder, practically easier)
- And many other approaches ...

Use c/c++!

Annoying things in c++:

- pointers
- memory leaking
- complicated paradigm
- building system
- using third-party packages

Use c/c++!

Annoying things in c++:

- pointers $\rightarrow$ you **don't need** to use pointers in contest
- memory leaking $\rightarrow$ you **don't need** dynamic memory allocation in contest
- complicated paradigm $\rightarrow$ you **only** need simple paradigm in contest
- building system $\rightarrow$ you **only** need to compile single file in contest
- using third-party packages $\rightarrow$ you can **only** use builtin libraries in contest

Use c/c++!

Annoying things in c++:

- pointers $\rightarrow$ you **don't need** to use pointers in contest
- memory leaking $\rightarrow$ you **don't need** dynamic memory allocation in contest
- complicated paradigm $\rightarrow$ you **only** need simple paradigm in contest
- building system $\rightarrow$ you **only** need to compile single file in contest
- using third-party packages $\rightarrow$ you can **only** use builtin libraries in contest

Rest parts of c/c++ are sweet!

Sweet c/c++: compile single file

TL;DR:

```
# minimum, get executable a.out
g++ myalgo.cc
# specify optimize level for debugging
g++ myalgo.cc -g -O0 -Wall -o myalgo
# specify optimize level for running
g++ myalgo.cc -O3 -o myalgo
```

- compiler: e.g. `gcc`, `g++`, `clang`, `clang++`
- path of source file: e.g. `myalgo.{c,cc,cpp}`
- argvs
  - `-g [-O0]`: for debugging, slow but more conservative
  - `-O[0/1/2/3]`: for running, fast and well optimized
  - `-o`: set executable name, default is `a.out`

Sweet c/c++: no malloc

```
1  // N is the upper bound given by problem statement
2  const int N = 100;
3
4  // use static memory and int index
5  int arr[N], cnt;
6
7  cnt = 0;                        // "clean" memory
8  for (int i=0; i<m; i++) {
9    arr[cnt++] = dummyelem  // add element
10 }
```

Sweet c/c++: no pointers

```
1  // stack
2  int stack[N], cnts = 0;       // clear
3  stack[cnts++] = dummy_elem; // push stack
4  dummy_elem = stack[--cnts]; // pop  stack
5
6  // queue
7  int que[N], front, tail;
8  front = tail = 0;             // clear
9  que[tail++] = dummy_elem;    // push queue
10 dummy_elem = que[front++];   // pop queue
11
12 // circular queue
13 tail  = (tail + 1) % size    // when push
14 front = (front + 1) % size   // when pop
```

Sweet c/c++: no pointers

```
1  // sort
2  int arr[N], n = 10;
3
4  // sort arr[0:n-1]
5  sort(arr, arr+n);
6
7  // sort with specified compare function
8  bool cmp(int x, int y) { return x < y; }
9  sort(arr, arr+n, cmp);
```

Sweet c/c++: looping

Very important:

- **Pre-condition:** what should be true before the loop;
- **Invariant:** what should be maintained by the loop;
- **Post-condition:** what should be true after the loop;

Most of bugs are caused by mistakes on them.

Example

In the problem **Fast ad Furious**, you decide to simulate the motion based on the brute-force thinking (on shortest path and change direction when necessary). But note that changing direction may cause the agent away from the shortest path, and thus break your **invariant**.

Try to apply these concepts to explain why pointer-free stack, queue and circular queue work (previous page).

Sweet c/c++: looping

```
1  // for-loop
2  for (<pre-cond>; <invariant>; <action>) { <actions> }
3  // example:
4  for (int i=0; i<n; i++) {
5    que[tail++] = dummy_elem;
6  }
7  // another example:
8  int i = 0;
9  for ( ; ; ) {
10   if (i >= n) break;
11   que[tail++] = dummy_elem;
12   i++;
13 }
```

There are also **while-loop** and **do-while** in c/c++, but they are interchangeable with **for-loop**.

Sweet c/c++: standard IO

C/C++ input is stream-based which will tokenize the content, while Python is line-based.
Example:
For input text `"Year 2020\n"`, c/c++ regard it as `["Year", "2020"]`; while Python will read the whole line, and further parsing is required.

```
// stream-based input
char str[N]; int year;
scanf("%s %d", str, &year);
// get whole line
char line[N];
gets(line); // this is unsafe in practice, but fine in contest
```

Resources

- C/C++ IO - `https://vjudge.net/contest/361790`
- Training contest - `https://vjudge.net/contest/383886`
- Extra resources for problem solving:
  - GCJ Round-1 `https://codingcompetitions.withgoogle.com/codejam/archive`
  - Google Kick-start `https://codingcompetitions.withgoogle.com/kickstart/archive`