COMP319 Algorithms 1, Fall 2021
Homework Programming Assignment 5 (HW5)
알고리즘1, 2021년 가을 학기, 프로그래밍 숙제 5

Instructor: Gil-Jin Jang Email: gjang@knu.ac.kr
School of Electronics Engineering, Kyungpook National University
장길진, 경북대학교 전자공학부

Objective

1. Understand shortest path algorithms and apply them to real problems.
2. Find shortest path given constraints.

Common Requirements

ID and name write your ID and name in a comment block at the beginning of the code.

Citation shortest path algorithm implementations are already given in the lecture slide. Moreover, there are huge number of shortest path algorithm implementations that are available on the Internet. Therefore, if you make use of them, there is a high chance that your code could be almost identical to that of other students. Even if you do not refer to any other source, your code may eventually become very similar to each other because you have to implement the same algorithms. To avoid any unwanted COPY, if you have referred to any material, write the web address, name of the textbook, lecture number and page number, etc., as comments near the relevant code lines. There is no reduction in scores by using publicly available codes.

Character code use standard ASCII characters only. Special characters or language-specific symbols can take more than 1 byte, and may cause compilation errors. There is no explicit penalty for using special characters, but students should be responsible for the compilation errors due to those.

Execution time the execution time of your code can be measured by using a built-in function `clock()` defined in `time.h`. In the given template file, two static functions, `reset_timer()` and `elapsed_time_in_sec()` are given, and you can measure the time of your code by placing `reset_timer()` in the beginning and `elapsed_time_in_sec()` at the point that you want to measure the time from when the `reset_timer()` is called.

```
reset_timer();    // reset the start time
....              // statements to measure time
t = elapsed_time_in_sec();
            // time in seconds from when reset_timer() was called
```

Memory usage the amount of dynamic memory use can be measured by the provided `malloc_c` and `strdup_c` functions. The number of bytes allocated by these functions are accumulated to a static variable `used_memory` so that we should figure out how much memory is required for your algorithm given a specific input. See the provided template code for detailed implementation.

Redefined memory/string functions all the dynamic-sized arrays should be allocated by `void *malloc_c(size_t)`, and string duplication should be done by `char *strdup_c(const char*)`. These functions replace the built-in `malloc` and `strdup` to count the allocated number of bytes.

Allowed memory operations Ordinary variables and fixed-sized arrays. The amount of memory for the above cases will be ignored in the measurement, because their sizes are not affected by the input size.

Allowed string functions almost all string functions except `strdup` are allowed, such as `strlen`, `strcmp`, `strncmp`, `strcpy`, `strncpy`, `strcat`, `strncat`, etc. You can also use `strtok`, but most of your homework assignments can be done without it. Try not to use it.

```
// ALLOWED USAGE EXAMPLES
int i, j, num_items;
char buf[1024];
char *s1, *s2;
struct container *C1, *C2;

s1 = strdup_c(buf);
s2 = (char*)malloc_c(sizeof(char)*(strlen(s1)+1));
strcpy(s2,s1);
if ( strcmp(s1, s2) == 0 ) {
   ...
}
C1 = (struct container*)malloc_c(sizeof(struct container)*num_items);
```

Unallowed functions `strdup` and `malloc`, and most memory manipulation functions such as `calloc`, `realloc`, `memcpy`, `memccpy`, `memmove`, `wmemmove` or other direct memory copy/move functions. These functions can update a large number of bytes at a single execution cycle so that memory move/copy operations should be efficiently implemented, therefore break UNIT TIME OPERATION assumption in the algorithm design.

```
// UNALLOWED USAGE EXAMPLES
s1 = strdup(buf);
C1 = (struct container*)malloc(sizeof(struct container)*num_items);

// some compilers may allow this, but do not use in your homework
char s2[num_items];
// direct memory copy (hardware acceleration may help)
memcpy(C2, C1, sizeof(struct container)*num_items);
```

---

1 Homework 5

A salesperson travels over cities using an airplane. Given the map of cities, start and final cities, and limited amount of fuel, find a path that visits most number of cities. Constraint: **on a 2-D space, forward movement is allowed along x-axis.**

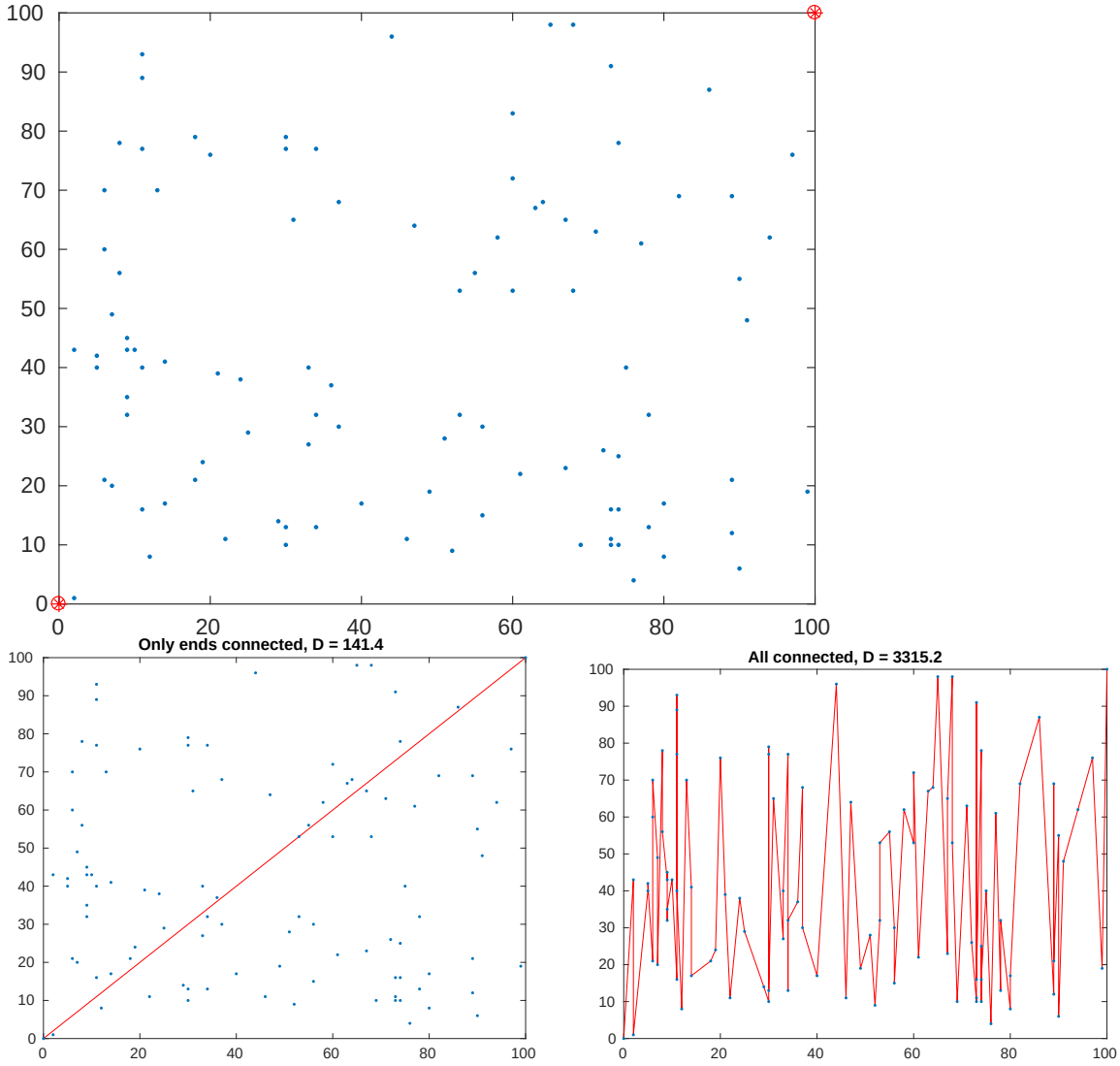
Note: due to the last constraint, this is not the well-known Traveling Salesperson Problem (TSP).

Input File Format 1. (Fixed) a map of 100×100 units

2. (Fixed) start city coordinate is $(0, 0)$, and destination city coordinate $(100, 100)$.

3. The (x, y) coordinates of other cities, $1 \leq x \leq 99$, $1 \leq y \leq 99$ (note: not sorted)

```
100                 (number of cities, fixed to 100)
0 0                 (first city's x y coordinate)
25 75  8 39  48 2  66 85  43 69  56 17 ...
( x y coordinates of cities 2 ~ 99. See the attached input file)
100 100                 (x y coordinate of the last city, fixed)
```



(2-D visualization of an input file. The left one is connecting start city and destination city. The right one shows a path with satisfying the constraint, "forward-only along x-axis")

Output 1. There are 4 different amounts of fuel — **250, 500, 950, 1350** —, and find the paths for the 4 amounts. Assume 1 fuel unit = 1 distance unit.

2. Print: given fuel (distance), actually traveled distance, the coordinates of the visited cities

3. Example output (not an solution, just to show the display example. ... are abbreviation).

250 249.3 (The maximum distance less than or equal to 250. 1 digit after decimal point)
0 0 3 21 ... 100 100 (x y coordinates of intermediate cities)
__._____ seconds (Execution time)
_____ bytes (Memory usage)

500 497.9 (The maximum distance less than or equal to 500)
0 0 8 39 9 25 ... 100 100 (x y coordinates of intermediate cities)
__._____ seconds
_____ bytes

950 949.9
0 0 2 76 9 5 ... 98 56 100 100 (x y coordinates of intermediate cities)
__._____ seconds
_____ bytes

1350 1347.2
0 0 2 43 5 40 6 21 ... 97 76 100 100 (x y coordinates of intermediate cities)
__._____ seconds
_____ bytes

NOTE:
1. Forward only along x-axis
2. The distance between (x_1, y_1), (x_2, y_2) is computed by Euclidean distance,
$$d\left((x_1, y_1), (x_2, y_2)\right) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$
3. There are cities with the same x coordinate. In that case, you can visit them in any order between them.
4. Both x and y can be the same — consider them as distance 0. So visiting one of them is equivalent to visiting all of them.

Attached:
1. No template code
2. Example input files: `hw5input.zip`

Evaluation Scheme

The total score of homework 5 is 100, and 20% of the whole course evaluation.

10pts basic submission score

10pts ID/name exist and are correct

30pts no compilation error and correct execution result.

- Will be evaluated by unknown examples.
- All real numbers should be displayed by 1-digit after the decimal point (ex: 1.0)
- Line break differences are allowed. Use single space (‘ ’) between items in the same line.
- Compilation error ⇒ Not executable ⇒ 0 points
- Score deduction if not implemented properly, for example, using wrong algorithm.

10pts Execution time

- 10: similar to other students or not too much slow
- 0: too slow

10pts Memory usage

- 10: similar to other students or not too much slow
- 0: too slow

Deduction for wrongful insertion of memory usage measurement code.

30pts code and report reading scores (requirements, etc.)

X 0 COPY makes whole scores 0. COPIED/BEING COPIED same. Leave citations as much as possible.

Submission format

Files to submit Source files only. Up to 10pts deduction when unnecessary files are included (input files, execution files, project files, etc.)

Submission 1. make a zip file `hw5.zip` of `hw5.c` and `hw5.pdf`.

hw5.pdf explain your algorithm briefly in A4 1 page. Included in code reading score.

2. upload it to `lms.knu.ac.kr`. The LMS system changes the submitted files, and this zip file submissoin is to preserve the source file name.

Due Friday 12/10 23:59 LMS time

Late submission Saturday 12/11 09:59 LMS time, -10pts per hour