CMSC 3613

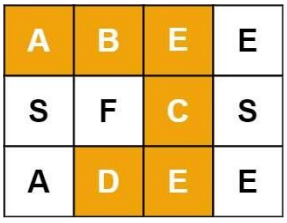
Programming Assignment: Backtracking

Due Date:

Check the D2L Calendar for the due date.

Assignment:

Write a program that uses backtracking to solve the following problem.



We need to perform word search in an m×n grid of upper-case letters. For example, we have a grid as showed in the figure above. We want to search the word "ABECED" in this grid. The search can only go horizontally or vertically. The matched sequence is highlighted. Once we find a matched result, we generate a message "Solution found!" This message has been implemented in the framework of this project. Your tasks are marked as TODOs in the comments of the given code, and please do not change the pre-defined function headlines.

1. Read the input, and this is the TODO 1 part in the given code. The grid and target word can be specified in an input file. The format of the input file (e.g., input.dat) will be similar as:

   ```
   ABECED
   3 4
   A B E E
   S F C S
   A D E E
   ```

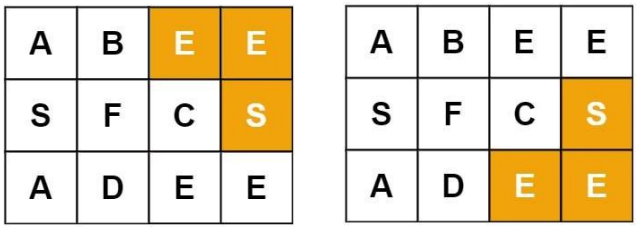
   {The first line is the target word. The second line specifies row number and column number in the grid. Starting from the third line, the grid will be specified. Your program should validate the input format, e.g., the grid and the row/column numbers are consistent, the data types are correct, etc.}

2. Search the target word in the grid, and this is the TODO 2 part in the given code. Please pay attention to the `const char* target` part in the headline of this function: we do not have an index for the target word to be passed on during the recursive search procedure, instead we use the char pointer to serve this purpose. This is a key point for this task.

   Please **note**: the same letter cell may not be used more than once. E.g., the target word "SEEE" can not be found in this grid above.

Bonus tasks:

1. Once we find a match, we need to output the sequence of cells in this grid, based on the coordinates of the cells. The coordinates are in the format of `(row, col)`, where `row` and `col` are row number and column number respectively, starting from 0. In this example of searching "ABECED", the sequence of cells should be:
   `(0,0) (0,1) (0,2) (1,2) (2,2) (2,1).`
   There could be several matched results. For example, if we want to search "SEE" in the grid above, there will be two matched results:



   So there will be two sequences as the output:
   `(1,3) (0,3) (0,2)` and `(1,3) (2,3) (2,2).`
   The output sequence can be specified in an output file. The format of the output file (e.g., output.dat) will be similar as:

```
(1,3) (0,3) (0,2)
(1,3) (2,3) (2,2)
```

   If there is no matched result, just keep the output file empty.

2. Can you try to avoid using a "global" mask for revisit such as a 2D bool array, i.e., do the search without explicitly using extra intermediate data structures (except the call stack of recursion and one structure for the result)? Please **note**: the same letter cell may not be used more than once in search.

Requirements:

1. Your program should follow similar backtracking framework discussed in the lecture, which is given as follows:

```
solve_from (Queens configuration) {
  if configuration already contains eight queens
    Print configuration
  else {
    for every square p that is unguarded by configuration {
      add a queen on square p to configuration;
      solve_from(configuration);
      remove the queen from square p of configuration;
   }
    }
}
```

Please provide your code for the TODOs marked in the given source file. Please do not change the pre-defined function headlines, and pay attention to the underscored specification in this document.

2. Your implementation should find a solution for small input within a reasonable time period.

3. The file of the input data should be provided in command line argument as follows:
   p02 input.dat output.dat
   p02 is the name of your executable (p02 should be a fixed name!), and input.dat, output.dat are the names of the input file and output file (the input/output names can be any). The provided code is specifically implemented for this format.

Evaluation:

This project will be evaluated according to the correctness of the various tree functions specified above, and secondarily according to the quality of your code. The rubric is as follows:

Categories		Weights	
Task 1		20%	
	Read target word		5%
	Read the grid		10%
	Error handling		5%
Task 2		60%	
	Backtracking framework		20%
	Base case		10%
	Recursive call		20%
	No revisit to letters		10%
Report		20%	
	3 questions in report item 4)		5% * 3
	The rest		5%
Bonus Task 1		30%	
Bonus Task 2		5%	
Coding style bonus		10%	

Submission:

1. Please provide a readme file, to help with the compilation and execution of your code. For example, information about your operating system and detailed command to compile your code should be included.

2. You need to submit a report by the due date. The report should include the following items:
   1) Your name and UCO email address
   2) The project number, i.e., p02
   3) A brief discussion of your implementation: just like an explanation of your idea in an interview. Please also brief explain:

- Why there is a nested loop in the main function to initiate the search procedure, compared to other examples of backtracking in the lectures?
- What's the purpose of using pointers and references in the function headlines, e.g., `string& target`, `const char* target`, `Grid<char>* grid`?
- How would you like to handle `const char* target` part in the function of `bool search(Grid<char>* grid, int r, int c, const char* target)`?

4) A screenshot of a test run.

3. Your source code also needs to submitted.

4. All the files need to be zipped as **p02_group*.zip**, where * means your group number, e.g., the submission from group 3 should be named as p02_group3.zip.

Notes:

1. To be considered on time, the program must be turned in by the due date.
2. Programs must reflect your knowledge and work, not others. You may ask others questions about algorithms, methods and programming style, but when you start writing code, you must work only with your group member(s).
3. Program grades reflect both the performance of the program and the programming style of the program.