Outline

Last time:

- Review of asymptotic analysis
- Empirical performance evaluation
- Review of foundational data structures

Today:

- Review of basic ADTs
- PA1 Walk Through
Abstract Data Type

Abstract Data Type (ADT): a higher-level data representation that helps us conceptualize and manipulate a problem (at a higher-level than array and linked-list)

“(Almost) any problem in computer science can be solved by another layer of representation”

Object-oriented ADTs come with pre-defined interfaces

How ADTs are implemented is hidden from the users
Vector ADT

What is a vector?

How is a vector different from an array?

Interfaces: isempty(), size(), ith_el(), insert_at_i(), replace_i(), delete_i(), etc.

Possible implementation: as an array
Queue ADT

What is a queue?

Example usage:
Queue ADT

What is a queue?
A “line” of items with FIFO access: the first item into the queue is the first one out

Example usage: line at the bank, line at the bus stop

Operations on queue (Fig. 6.4):
enqueue(), dequeue(), head(), create(), size()
Queue ADT Implementation

As a circular array (Fig. 6.5):

- head, tail indices to keep track of front and end of line
- keep a count to disambiguate a full queue from an empty queue
- use modulo array length to wrap around at the end of array

As a linked list:

- enqueue(): add to tail
- dequeue(): remove from head
- head(): element head points to
Example Queueing Problems in CS

Job scheduling in CPU

Packet scheduling in network router

Usual queueing diagram:

\[ \mu \]
Deque ADT

Deque (pronounced “deck”): double-ended queue

Items can be inserted and removed from both ends of the deque (Fig. 6.8)

Operations: head(), enqueue_head(), dequeue_head(),
tail(), enqueue_tail(), dequeue_tail(), create(), size(), etc.

Implementation (Fig. 6.9): as a doubly-linked list

Sugih Jamin (jamin@eecs.umich.edu)
Stack ADT

What is a stack?

Example usage:
Stack ADT

What is a stack?
A “pile” of items where new objects are put on top of the pile and the top object is removed first (LIFO)

Example usage: cafeteria tray, a pile of paper napkins, CD-ROM on spindle

Operations on stack (Fig. 6.2):
push(), pop(), top(), create(), size()
Stack ADT Implementation

As an array:

- `push()`: add item to end of array, $O(1)$
- `pop()`: remove item from end of array, $O(1)$
- `top()`: look at item at end of array, $O(1)$

As a linked-list:

- `push()`: append (prepend) item, $O(1)$
- `pop()`: remove item from tail (head), $O(1)$
- `top()`: look at last (first) item, $O(1)$
Example Usage of Stack: RPN Calculator

Usual math (infix) notation: $5 + 9 \times 2 + 6 \times 5$
Can you use a stack to implement a calculator?

Order of operations is determined by *operator precedence*, parentheses used to override precedence rules: $(5 + 9) \times 2 + 6 \times 5$

Polish (prefix) notation does not require operator precedence rules, therefore does not require use of parentheses: $+ \times + 5 9 2 \times 6 5$

Or equivalently, Reverse-Polish (postfix) Notation:
$5 9 + 2 \times 6 5 \times +$

Now the calculator can be implemented on a stack (Fig. 6.3)
(Reverse) Polish notation is a more minimalist notation (in math, minimal == elegant) However, parentheses may be used as syntactic sugar, i.e., eye candy, to ease legibility, but not required for correctness

So “+ * + 5 9 2 * 6 5” can be written as “+(+(5, 9), 2), *(6, 5))”
What does this look like?
Example Usage of Stack: Function Call Stack

How is the function call stack implemented?

Push formal arguments on stack, call function, push local variables on stack, pop stack on return from function, push return value on stack

See Figs. A.8, A.10 from Patterson&Hennesy
Buffer Overflow Attack

Further reading (optional):
*Smashing the Stack for Fun and Profit* by Aleph One, http://www.phrack.org/show.php?p=49&a=14

Try this at home, but NOT on the Internet!
(Already done: Internet Worm 1988)
PA1: Programming Assignment 1

Topic: path finding (one of the primary areas of game programming)

Due date: 10/4, 1 PM

Report due: 10/9, 1:40 pm in class

To be done individually

No STL

Always get the latest revision of the spec
The Game

Rescue a victim from a building, or defuse a bomb in the building, whichever of the victim or the bomb is closer.

The Building: $n$ levels, $0 \leq n \leq 9$

Each level is an $M \times M$ square, $M \geq 4$

Each level can have different dimension and orientation.
Input File

version=281F07.i
nlevels=1
level=1
dimension=8
# this is a comment
map=
Vxxxxxxxx
 x     B
 x  x  xxx
 x  x  x
 x  xx
 x  xx
 x    x
 x     Sx
xxxx2xxx
Locations

Starting position: ‘S’

Bomb location: ‘B’

Victim location: ‘V’

The target: the closer of the bomb or victim

All case sensitive
Tile Types

Open tiles: ‘ ‘

Walls: ‘x’

You cannot walk through walls

Some walls may have crumbled, if you step off the $M \times M$ grid of each level, you have a long fall in your grade

Digits (0-9): staircases

Rubbles: ‘r’
Staircases

Each level can be connected to another level by a staircase.

Staircases are bidirectional: if both levels exist, the staircase connecting them must exist on both levels.

It is ok for a level to be inaccessible.

It is ok for a staircase to lead to a non-existent level.

Two levels cannot be *directly* connected by more than one staircase.
Directions

Top of file is north

Direction markers:
n: north; s: south; e: east; w: west;
h: northwest; j: southwest; k: northeast; l: southeast

Most northwesterly tile is coordinate (0,0)

Staircases do not have to start and end at the same coordinate at both levels

Sugih Jamin (jamin@eecs.umich.edu)
**Path Marking**

Starting from the tile adjacent to the ‘S’ tart position, mark each tile with the direction taken in the *previous* step.

Overwrite the target tile with the direction of the final step taken.

Upon stepping on a staircase tile, overwrite the staircase tile with the direction of the previous step taken.

If the stairs are taken, do not overwrite the staircase tile on the **exit** level.

When standing on a staircase tile, all adjacent tiles on both levels are visible.

Sugih Jamin (jamin@eecs.umich.edu)
Output File

# output for file test.txt
version=281F07.o
path_cost=7
nlevels=1
level=1
dimension=8
map=
Vxxxxxxx
x   kee
x   xnxxx
x   x hx
x   xxn
x   h x
x   Sx
xxxx2xxx
Map Representation

Each level can be stored as a multi-dimensional array

A multi-dimensional array must be implemented as a single contiguous chunk of memory (you can call `new` only once, though you may use other helper structures to access the array)

As you read in the map, remember where the staircases at each level are

You are *not* to record the location of the bomb and victim for direct access, i.e., you are to search for them on the map

Sugih Jamin (jamin@eecs.umich.edu)
Path Finding

Implement three algorithms:

1. queue based
2. stack based
3. variable-cost algorithm
Queue-based Algorithm

enqueue start position

loop {
    dequeue next tile
    enqueue all unique adjacent tiles
} until target reached or queue is empty

if (queue is empty) search failed

if (target reached) show path taken

Pick one of multiple possible paths
Stack-based Algorithm

Stack-based: same as before, except use push and pop instead of enqueue and dequeue

One of queue-or stack-based must find the path with the lowest cost (least number of tiles traversed)

Do NOT discuss or post on the forum which one is the better data structure/algorithm
Variable-cost Algorithm

Different tile types now have different costs to move into:

- an open tile, the bomb, or the victim costs 1
- a rubble-filled tile costs 5
- a staircase tile but staying on same level costs 2
- a staircase tile and taking the stairs between levels $i$ and $j$ costs $(2 + |level_i - level_j|)$
- wall tiles are still impassable ($\infty$ cost)

Adapt the more optimal of your queue- or stack-based algorithm to find lowest cost path with the above cost function and report the resulting path_cost according to the same cost function
Timing Your Code

Measure only the time it takes your path-finding algorithm to run, don’t include time to do I/O. For example:

```c
init_stuff();
read_map();

gettimeofday(&start,0);
path_find();
gettimeofday(&end,0);
timing = end-start; /* PSEUDO-CODE! */
report(timing);
```

Time all three algorithms, for $N$ different-sized maps, then plot the results using gnuplot.
Command Line Options

-\texttt{-q}: run queue-based algorithm
-\texttt{-s}: run stack-based algorithm
-\texttt{-v}: run variable-cost algorithm
-\texttt{-i <string>}: take string as input file name, default stdin
-\texttt{-o <string>}: take string as output file name, default stdout
-\texttt{-e}: run timing analysis, must be used with one of \texttt{-q}, \texttt{-s}, or \texttt{-v}
-\texttt{-h}: help

-\texttt{-q}, \texttt{-s}, and \texttt{-v} are mutually exclusive

How do you read the command line options?
Never seen \texttt{main(argc, argv)}?
Error Codes

If you run into an error, print out an error message and call `exit()` with the appropriate error code:

- EBADOPT -10
- EBADFILE -20
- EBADVERSION -21
- EBADFTYPE -22
- EBADMAP -30
- EMULTS -31
- EMULTB -32
- EMULTV -33
- EMULTC -34
- ENOSTART -35
- ENOBOMB -36
- ENOVIC -37
Exit Codes

Upon correct exit, also call exit() with the appropriate exit code:

NOERR    0
NOPATH   20
PATHTOB  21
PATHTOV  22
Other Items

Coding style

Empirical efficiency

Start immediately

Don’t debug with cout/printf’s

Help stop 48 hours to due date

Testing your code (may post test cases and outputs)

Your code MUST run on CAEN Linux machine, don’t wait till last minute/day to port, best to work on the system from the start

The autograder is NOT a debugger

Grade composition

What to turn in
Reminder: Course Grading Policy

Course grade composition:

- Final: 20%
- Midterm: 20%
- Homeworks (HWs): 12%
- Programming Assignments (PAs): 46%
- Class Participation: 2%

A total of four free late days (incl. weekends and holidays)

You must keep track of your own late days
Cheating and Collaboration

You are encouraged to learn from each other.

We look forward to lively discussions in class and on the forum.

However, cheating is not tolerated and will be reported to the Honor Council.

Cheating is when you copy, with or without modification, someone else’s work.

It is also considered cheating to knowingly expose your solutions or to use some other publicly available solutions.

**It is also considered cheating to turn in one algorithm as another; be careful with cut and paste place holder.**

When in doubt, ask Sugih Jamin (jamin@eecs.umich.edu).