CS35X: Competitive Programming

Lecture 7: Stacks, Queues, Graphs

Joshua Brody

Warmup Kattis Problem:
relocation

Problem debrief: trianglesofasquare

Stack, Queue ADTs

- Maintain ordered collection of items.
- Main Stack Operations:
  - **push(foo)** — add foo to top of stack
  - **pop()** — remove top element of stack
  - Items accessed in **LIFO** order
- Main Queue Operations:
  - **enqueue(foo)** — add foo to back of Queue
  - **dequeue()** — remove element from front of queue
  - Items accessed in **FIFO** order.
- Other Stack/Queue operation(s): ***isEmpty(), getSize()***
- Interfaces are ***simple***. Operations should be ***fast — O(1)!***

STL deque class

- Double-ended queue — add/remove items from front or back.

```
#include <deque>                   // include pair library
deque<string> d;                   // d is deque of strings
deque<int> q = {3,2,6};            // direct initialization
q.push_front(8);                   // q is now [8,3,2,6]
q.push_back(9);                    // q is now [8,3,2,6,9]
q.pop_front();                     // q is now [3,2,6,9]
q.pop_back();                      // q is now [3,2,6]
q.front();                         // returns 3
q.back();                          // returns 6
q.empty();                         // returns false (q not empty)
```

Implementation Details

- Deque (**d**ouble-**e**nded **que**ues) built like *circular* ArrayLists.
- Adding/removing to front/end of deque is O(1) in practice.
- **pop_front, pop_back** don't return the element.

Stack/Queue application: Graphs

- A graph **G = (V,E)** is a set of *vertices* **V** along with a set of *edges* **E**.
- Graphs represent binary relationships.
  - Map: vertices == towns, edges == roads
  - Social network: vertices == users, edges == friendships
  - Temporal network: vertices == events, edge (u,v): u happens before v.
- There are many interesting algorithms we can do on graphs

Graph Representation: Adjacency Lists

- Say graph has **n** vertices 0,…, n-1.
- Represent graph **g** as array of *vector<int>*:
    - **g[i]**: vector of neighbors of i.

A first Graph algorithm: BFS

- Visit all nodes in graph, using queue to manage exploration
- `deque<int> q;`
- `q.push_back(s);`
- `visited[s] = true;`
- `while(!q.empty()) {`
  - `v = q[0];`
  - `q.pop_front();`
  - `for(int i=0; i<g[v].size(); i++) {`
    - `u = g[v][i];`
    - `if(! visited[u]) {`
      - `// visit u`
      - `q.push_back(u);`
    - `}`
  - `}`
- `}`

Things we can do with BFS:

- Are all vertices connected?
- Find shortest length path from s—>t.
- Identify vertices reachable from s.
- Count # connected *components* of graph
- …

Graph Implementation tips

- Represent vertices of graph as integers.
- Adjacency List can be dictionary instead of array, but **huge** time penalty.
- Adjacency List can store:
  - Vertex neighbors
  - Edges out of a vertex
- Create a helper function that loads graph into an adj list
- Consider defining a class to represent *weighted* edges.
- String vertices:
  - Store Dictionary mapping integers to (string) vertex label
  - Run graph algorithm on graph using int vertices
  - Use dictionary to recover vertex names if needed

Kattis Problem: onaveragetheyrepurple