Ben Kahan
DS 210
Final Cheatsheet
20 December 2022

Rust

Rust Books:

Rust Docs: `rustup docs --book`

Rust By Example:

```
cd /rust-by-example
mdbook serve
```

Go to `localhost:3000` to view the book.

Graph Algorithms

From Wikipedia: Using a Priority Queue

Dijkstra's Algorithm

- Worst case: O(E + V*log(V)), using Fibonacci Heap min-priority queue

```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                           // Initialization
3
4      create vertex priority queue Q
5
6      for each vertex v in Graph.Vertices:
7          if v != source
8              dist[v] ← INFINITY                 // Unknown distance from source to v
9              prev[v] ← UNDEFINED                // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                      // The main loop
15         u ← Q.extract_min()                    // Remove and return best vertex
16         for each neighbor v of u:              // Go through all v neighbors of u
```

```
17              alt ← dist[u] + Graph.Edges(u, v)
18              if alt < dist[v]:
19                  dist[v] ← alt
20                  prev[v] ← u
21                  Q.decrease_priority(v, alt)
22
23      return dist, prev
```

BFS

- Vertex Based
- Queue data structure
- O(V + E) for adj_list
- O(V^2) for adj_matrix

```
Input: A graph G and a starting vertex root of G

Output: Goal state. The parent links trace the shortest path back to root[8]
```

```
 1  procedure BFS(G, root) is
 2      let Q be a queue
 3      label root as explored
 4      Q.enqueue(root)
 5      while Q is not empty do
 6          v := Q.dequeue()
 7          if v is the goal then
 8              return v
 9          for all edges from v to w in G.adjacentEdges(v) do
10              if w is not labeled as explored then
11                  label w as explored
12                  w.parent := v
13                  Q.enqueue(w)
```

DPS

- Edge based
- Generally recursive in nature (anything recursive can be iterative and vice versa)
- Uses the callstack or stack data structure
- Same time complexity as BFS

Recursive Implementation:

```
procedure DFS(G, v) is
    label v as discovered
    for all directed edges from v to w that are in G.adjacentEdges(v) do
        if vertex w is not labeled as discovered then
            recursively call DFS(G, w)
```

Iterative Implementation:

```
procedure DFS_iterative(G, v) is
    let S be a stack
    S.push(v)
    while S is not empty do
        v = S.pop()
        if v is not labeled as discovered then
            label v as discovered
            for all edges from v to w in G.adjacentEdges(v) do
                S.push(w)
```

Krustal's Algorithm for MWSF

- Finds minimum weight spanning forest for undirected edge-weighted graph

```
algorithm Kruskal(G) is
    F:= {}
    for each v in G.V do
        MAKE-SET(v)
    for each (u, v) in G.E ordered by weight(u, v), increasing do
        if FIND-SET(u) != FIND-SET(v) then
            F:= F U {(u, v)} U {(v, u)}
            UNION(FIND-SET(u), FIND-SET(v))
    return F
```

Prim's Algorithm for MWST

```
1. Associate with each vertex v of the graph a number C[v] (the cheapest cost of a connectio
2. Initialize an empty forest F and a set Q of vertices that have not yet been included in F
3. Repeat the following steps until Q is empty:
    a. Find and remove a vertex v from Q having the minimum possible value of C[v]
    b. Add v to F
    c. Loop over the edges vw connecting v to other vertices w. For each such edge, if w sti
         i. Set C[w] to the cost of edge vw
        ii. Set E[w] to point to edge vw.
4. Return F
```