Analysis of Algorithms - Notes

Philip Warton

November 19, 2020

1 Graph Search Algorithms

We begin with this simple example of the "whatever first search". This is an algorithm that will brute force find some path from $v \to s$ for any s-reachble vertex v.

Whatever-first-search algorithm with path rememberance:

```
WFS(G, s) {
    Parent(s) = $\emptyset$
    Bag = $\{(s, \emptyset)\}$
    while Bag $\neq \emptyset$ {
        $(v, p)$ = any vertex from Bag
            (remove $v$ from Bag)
        if $v$ is not marked {
            mark $v$
            Parent(v) = p
            for all $(v, w) \in E$ {
              '' add $(w, v)$ to Bag
            }
        }
    }
}
```

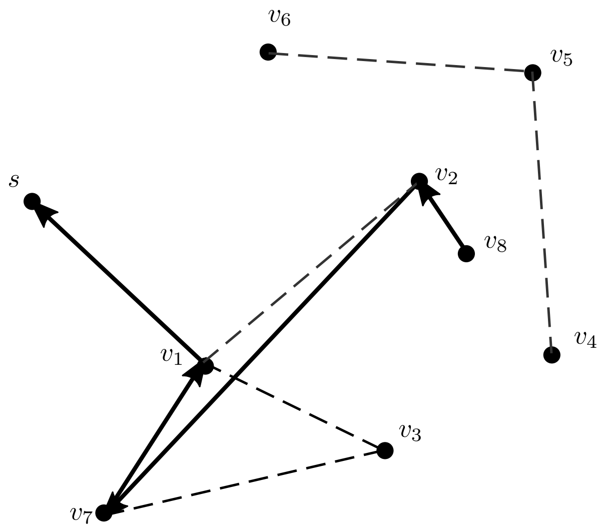

Figure 1: Whatever First Search For $v_8 \to s$

Once the bag is eventually empty, we can find the path from v to s by

$$v \to \text{parent}(v) \to \text{parent}(\text{parent}(v)) \to \cdots \to s$$

The WFS algorithm marks all vertices reachable from s.

Proof. Let s be our initial point. We use induction that is based on the shortest path s to v.

$\boxed{\text{Base Case:}}$ Our vertex $v = s$, and ShortestPathLength$(v \to s) = 0$, and WFS marks it on the first iteration.

$\boxed{\text{Inductive Step:}}$ For any point v for which the shortest path $s \leftarrow v$ is smaller than $k \in \mathbb{N}$, we assume that WFS has already marked v. Let v be a point for which its minimum distance from s is k. Then let u be the neighbor of v that lies on a shortest path from v to s. Then the length of $u \to s$ is $k - 1$, and by assumption u is marked. Since v is a neighbor of u, v will be marked as well. □

What kind of data structures would be good to use for Bag? If we use a stack, then this algorithm becomes a depth first search algorithm (DFS). If we use a queue, then we have a breadth first search (BFS) algorithm. If there is a weighted graph, one can use a priority queue based on edge weight, resulting in Dijkstra's shortest path algorithm.

```
Serach(G, s):
    Insert $s$ into Bag
    while Bag $\neq \emptyset$:
        $v$: any vertex from Bag
        if $v$ is not marked:
            mark $v$
            $\forall v \to w \in E$
                insert $w$ into Bag
```

If we use a stack, we have DFS, if we use queue we have BFS, if we have priority queue/heap, we have either Dijkstra's or Prim's algorithm.

Dijkstra's Algorithm

This algorithm utilizes a priority queue in order to find the shortest path from s to any vertex v.

```
Dijkstra(G, s):
    priQ = $\{(s, 0)\}$
    while priQ $\neq \emptyset$:
        $(v, d) \leftarrow$ priQ.ExtractMin
        if $v$ is not marked:
            mark $v$
            $\forall v \to w \in E$:
                priQ.Insert($w, d + l(v \to w)$)
```

The shortes path problem takes some directed graph $G = (V, E)$ where we have a length function $l : E \to \mathbb{R}^+$, $s \in V$. Then we output the shortest path from s to v for every vertex $v \in V$. The running time of this algorithm is $O(E \log V)$ since our priority queue insert

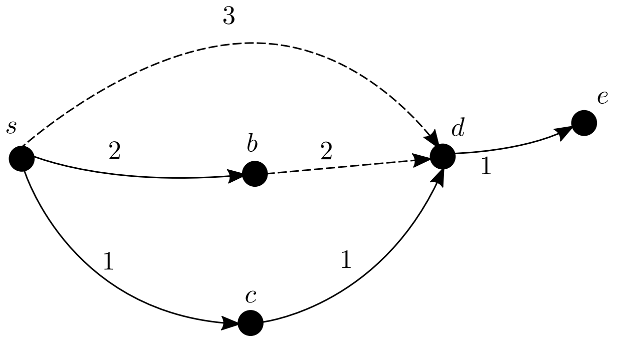

Figure 2: Dijkstra's Algorithm on $G = (V, E)$

function runs in $O(\log V)$ time. A graph with negative weights does not work, especially so if it is cyclic, because the shortest Dijkstra path may not be the shortest actual path, and if it is cyclic, there may not even exist a shortest path.

2 Minimum Spanning Tees

3 Greedy Algorithms

Our first example is a job scheduling algorithm. Each shift has a start time and an end time. The inputs for this algorithm are a list of start and end times, $S[1, 2, \cdots, n], F[1, 2, \cdots, n]$. The output will be the maximum number of disjoint shifts. The greedy algorithm

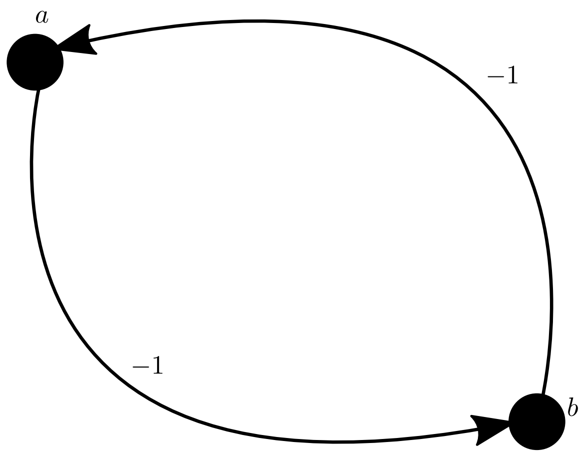

Figure 3: A Negative Cyclic Graph

solution, involves taking the largest start time, and go from there.

Greedy Algorithm to Solve the Job Scheduling Problem

```
JobScheduling($S, F$)
    Sort $F$, Permute $S$ accordingly
    endTime $= -\infty$
    todoList = $\{\}$
    for $i \leftarrow 1$ to n
        if $S[i] >$ endTime
            Add $i$ to todoList
            endtime = $F[i]$
    return todoList
```

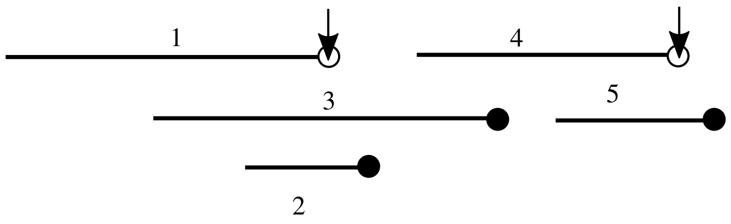

Figure 4: Job Scheduling Algorithm : Smallest Endpoint

The above algorithm does in fact return the maximum number of possible disjoint shifts.

Proof. Denote $g_1, g_2, \cdots, g_t$ as the output of our greedy algorithm, and denote $opt_1, opt_2, \cdots, opt_k$ to be the optimal solution with max shared prefix to our soluion. If $k \leqslant t$, then clearly g is optimal. Otherwise, $k > t$. Suppose that there exists $i \in 1, 2, \cdots t$ such that $g_i \neq opt_i$. Choose the first such i. Then for $n < i, g_i \cap g_n = \emptyset$. Then $F[g_i] \leqslant F[opt_i]$, and since opt_i is disjoint from all $opt_{i+1}, \cdots, opt_n$, it follows that g_i is disjoint with $opt_{i+1}, \cdots, opt_n$. Then $opt_1, \cdots, opt_{i-1}, g_i, opt_{i+1}, \cdots, opt_k$ is a valid scheduling and is optimal. Thus there exists some optimal solution with a prefix more similar to g than opt is, contradicting our definition of opt. □

Our next example will be the Huffman Coding problem. We want to make our file as small as possible. We must make the encoding prefix free, or if we think of it as a binary tree, every encoding must be a leaf node. Our input is the fequency of symbols, our output is the best possible prefix free code.

To do this, we can merge the two least frequent items $(x, f[x]), (y, f[y]) \rightarrow (`xy', f[x] + f[y])$. Then we compute the optimal tree with $n - 1$ symbols. Then we add the childred x, y to the leaf 'xy'.