CS170– Spring 2022— Homework 4

CurMack

August 9, 2022

Disjoint Set

Union by rank and path compresion:

Algorithm 1: `makeset`(x)

$\pi(x) \leftarrow x$
$rank(x) \leftarrow 0$

Algorithm 2: `find`(x)

if $x \neq \pi(x)$ **then**
    $\pi(x) \leftarrow$ `find`$(\pi(x))$

Algorithm 3: `union`(x, y)

$r_x \leftarrow$ `find`(x)
$r_y \leftarrow$ `find`(y)
if $r_x == r_y$ **then**
    return
if $rank(r_x) > rank(r_y)$ **then**
    $\pi(r_y) \leftarrow r_x$
else
    $\pi(r_x) \leftarrow r_y$
    if $rank(r_x) == rank(r_y)$ **then**
        $rank(r_y) \leftarrow rank(r_y) + 1$

Each `find` takes $\mathcal{O}(\log^* n)$ time.

Kruskal's algorithm

Kruskal's minimum spanning tree algorithm starts with the empty graph and then selects edges from E according to the following rule:

Repeatedly add the next lightest edge that doesn't produce a cycle.

Cut property: Suppose edges X are part of a minimum spanning tree of $G = (V, E)$. Pick any subset of nodes S for which X does not cross between S and $V - S$, and let e be the lightest edge across this partition. Then $X \cup \{e\}$ is part of some **MST**.

Algorithm 4: `kruskal`(G, w)

Input: A connected undirected graph $G = (V, E)$ with edge weights w_e
Output: A minimum spanning tree defined by the edges X

for *all* $u \in V$ **do**
  `makeset`(u)
$X \leftarrow \phi$
Sort the edges E by weight
for *all edges* $(u, v) \in E$*, in increasing order of weight* **do**
  if *find*$(u) \neq$ *find*(v) **then**
    add edge (u, v) to X
    `union`(u, v)

Prim's algorithm

Prim's minimum spanning tree algorithm:

Algorithm 5: `prim`(G, w)

Input: A connected undirected graph $G = (V, E)$ with edge weights w_e
Output: A minimum spanning tree defined by the edges X

for *all* $u \in V$ **do**
  $cost(u) \leftarrow \infty$
  $prev(u) \leftarrow$ `nil`
Pick any initial node u_0
$cost(u_0) \leftarrow 0$
$H \leftarrow$ `makequeue`(V) (priority queue, using cost-values as keys)
while H *is not empty* **do**
  $v \leftarrow$ `deletemin`(H)
  for *each* $(v, z) \in E$ **do**
    if $cost(z) > w(v, z)$ **then**
      $cost(z) \leftarrow x(v, z)$
      $prev(z) \leftarrow v$
      `decreasekey`(H, z)

Huffman encoding

Algorithm 6: `Huffman`(f)

Input: An array $f[1 \ldots n]$ of frequencies
Output: An encoding tree with n leaves

Let H be a priority queue of intergers, ordered by f
for $1 = i : n$ **do**
    `insert`(H, i)
for $k = n + 1 : 2n - 1$ **do**
    $i \leftarrow$ `deletemin`$(H), j \leftarrow$ `deletemin`(H)
    create a node numbered k with children j
    $f[k] \leftarrow f[i] + f[j]$
    `insert`(H, k)

Runtime:: $\mathcal{O}(n \log n)$ if a binary heap is used.

2 Updating a MST

(a) **Main Idea:** Do nothing.
Runtime: $\mathcal{O}(1)$ time.

(b) **Main Idea:** Add e to T. Use `DFS` to find the cycle that now exists in T. Remove the heaviest edge in the cycle from T.
Runtime: $\mathcal{O}(|V|)$ time.

(c) **Main Idea:** Do nothing.
Runtime: $\mathcal{O}(1)$ time.

(d) **Main Idea:** Delete e from T. Now T has two components, A and B. Find the lightest edge with one endpoint in each of A and B, and add this edge to T.
Runtime: $\mathcal{O}(|V| + |E|)$ time.

3 Twenty Questions

Main Idea: Creae a Huffman Tree on weights P.
Correctness: The Huffman tree gives us min code length. The code length is the cost of guess strategy, which means Huffman tree gives us min cost strategy.
Runtime: $\mathcal{O}(n \log n)$ (this is dominated by the time to sort the probabilities)

4 Graph Game

(a) Marking a node can only ever increase your score, since all values are positive.

(b) **Main Idea**:Picking nodes in decreasing order

Correctness:We have ordering $v_1, v_2, \ldots, v_n$ they are not sorted s.t. $l(v_i) < l(v_{i+1})$
case 1: v_i and v_{i+1} are disconnected. No edges between means the score doesn't change.
case 2: There are edges between v_i and v_{i+1} our score will increase by $l(vi+1) - l(v_i)$, if we swapped ordering such that $l(v_i) > l(vi+1)$. THe solution will be decreasing order.

Runtime:$\mathcal{O}(n \log n)$

(c) The same graph as example and take $l(A) = 1, l(B) = -1, l(C) = -2$. Then the greedy algorithm gives value 0. The optimum is A, B with value 1.

(d) The same graph as example and take $l(A) = 1, l(B) = -1, l(C) = -2$. Then the modified greedy algorithm gives A with value 0. The optimum is A, B with value 1.