CS180 Homework 3

Zhehao Wang 404380075 (Dis 1B)

Apr 16, 2016

1 Lexicographically smallest tree is a MST in an unweighted graph with distinct weights

Proof by contradiction: let T be the tree that our algorithm finds. Assume there exists a distinct MST T' which is lexicographically larger than T: T' has the same $w_1...w_{k-1}$ with T but a $w'_k > w_k$, and the sorted tuple $A' = (w_1, ..., w_{k-1}, w'_k, ..., w'_n)$.

Consider the graph T'' which is T' with the edge w_k included, there will be a loop. The loop contains at least one edge e such that $w(e)$ is w'_k, or comes after w'_k in array A: since if the loop only contains w_k and edges that come before w'_k in A', T would have a loop and contradicts with it being a spanning tree. Eliminate the edge e from T'', we break the loop and still have a spanning tree T'_s. Since $w(e) \geq w'_k > w_k$, the sum of weight of T'_s is smaller than that of T', which contradicts with T' being MST.

Thus there doesn't exist a MST who is not lexicographically smallest, and a lexicographically smallest tree is the MST. The conclusion holds.

2 MST defined as Dijkstra with max as evaluation criteria

(1)

Proof by contradiction: we have tree T that satisfies the description, assume that there's a distinct MST T' such that for some path (u, v), $max(\{w(e)|e \in path_{T'}(u, v)\}) > max(\{w(e)|e \in path_T(u, v)\})$, where $path_T(u, v)$ denotes a path from u to v in tree T.

Let edge (s, t) be an edge in path (u, v) in T', such that $w(s, t)$ is the $max(\{w(e)|e \in T', e \notin T\})$. There exists a $path_T(s, t)$, and $max(\{w(e)|e \in path_T(s, t)\}) < w(s, t)$. There exists at least one edge e' in $path_T(s, t)$ such that $e' \notin T'$ (or otherwise T' would have a circle), and $w(e') < w(s, t)$.

Consider T' with e' included, T' would have a circle involving nodes s and t, and this circle can be broken by removing the edge (s, t) from T'. The resulting graph T'' is still a tree, and has a smaller total weight than T', which contradicts with T' being the MST.

Thus there's not a distinct MST from our tree T, and T is the MST. The conclusion holds.

(2)

i. We want to prove that the resulting graph (as defined by the set of edges in each $P(v)$) has the same nodes, and is connected and acyclic.

Proof: according to the algorithm, all edges will be visited by the graph (assuming we have a connected graph). Since we add unvisited nodes to the queue upon visiting each edge and the algorithm only ends after the queue's empty, we will traverse all the nodes in the graph and for each node at least one path to reach it from s will be maintained. Thus the resulting graph is connected.

Assume a cycle $(s_1, ..., s_n, s_1)$ exists after the algorithm finishes. Let the last edge added to complete the cycle be $(s_k, s_{k+1 \mod n})$ where $k = 1..n$, thus the last dequeued node to complete this cycle is

$s_{k+1 \mod n}$. According to the algorithm, to put an edge (u, v) in the resulting graph, u should be dequeued, and each node is dequeued only once. Before dequeueing $s_{k+1 \mod n}$ and adding the edge $(s_k, s_{k+1 \mod n})$, edge $(s_{k+1 \mod n}, sk + 2 \mod n)$ is already in, so we have $s_{k+1 \mod n}$ is already dequeued. This contradicts with the nodes being dequeued only once. Thus the resulting graph is acyclic.

Since the resulting graph has the same nodes as the input graph, is connected and acyclic, we have that the resulting graph is a spanning tree of the original graph.

ii. The algorithm gives us a spanning tree T whose path from source s to any node t is bottleneck smaller than any other paths in G. We want to prove that T as given by the algorithm is MST, with a similar method as that of (1).

Proof by contradiction: assume that there's a distinct MST T' such that for some path (s, v), $path_{T'}(s, v)$ is bottleneck larger than $path_T(s, v)$, where $path_T(u, v)$ denotes a path from u to v in tree T.

Let edge (p, q) be an edge in path (s, v) in T', such that $w(p, q)$ is the $max(\{w(e)|e \in T', e \notin T\})$. There exists a $path_T(p, q)$, and $max(\{w(e)|e \in path_T(p, q)\}) < w(p, q)$: since if otherwise, let a be the maximum weight edge on $path_T(p, q)$ which satisfies $w(a) > w(p, q)$, a is in either $path_T(s, q)$ or $path_T(s, q)$; without loss of generality, let a be in $path_T(s, p)$, the tree T'' which is T with a removed and edge (p, q) added is still a spanning tree, and has a bottleneck smaller $path_{T''}(s, p)$ than $path_T(s, p)$, which contradicts with the definition of T.

Since there exists at least one edge e' in $path_T(p, q)$ such that $e' \notin T'$ (or otherwise T' would have a circle), and $w(e') < w(p, q)$. Consider T' with e' included, T' would have a circle involving nodes p and q, and this circle can be broken by removing the edge (p, q) from T'. The resulting graph T_s is still a tree, and has a smaller total weight than T', which contradicts with T' being the MST.

Thus we can find the MST by ensuring that the tree whose path from source s to any node t is bottleneck smaller than any other paths in G, and the algorithm is correct.

iii. The two MST algorithms construct the MST in the same order.

Proof by induction:

Base case: starting from the node s, let the first node dequeued by *DijkstraMST* be u. For the $(s, i) \in E$ where $i \in V$, (s, u) must have the smallest weight among all (s, i), since otherwise u won't be at the top of the priority queue. The smallest edge is also the first one chosen by *Prim*, thus the two algorithms have the same first edge, and the base case holds.

Induction hypothesis: the first n edges picked by both algorithms are the same.

Induction case: we want to prove that the $(n + 1)^{th}$ edge the two algorithms pick are the same.

Proof by contradiction: assume that *DijkstraMST* picks edge (u, v), while *Prim* picks (x, y), $v \neq y$. *DijkstraMST* dequeues v earlier than y, meaning that $P(v)$ is bottleneck smaller than $P(y)$. Sort the weights of paths in $P(v)$ and $P(y)$ in decreasing order, we have $P(v) = w_1, w_2, ..., w_m$, $P(y) = w'_1, w'_2, ..., w'_m$, and $w_k < w'_k$ for the smallest index k in $P(v)$ where $w_k \neq w'_k$. By induction hypothesis, we have that the first n edges picked by both algorithms are the same, since $w_k \neq w'_k$ and we have a distinct weight graph, either w_k is the edge (u, v), or w'_k is the edge (x, y).

In the case $w_k = w(u, v)$, *Prim* picked (x, y) because $w(x, y) < w(u, v)$. So for *DijkstraMST*, $P(y)$ is bottleneck smaller than $P(v)$, because for $P(v)$, its first difference $w_k = w(u, v)$ can be replaced by a smaller $w(x, y)$. This contradicts with $P(v)$ being bottleneck smaller than $P(y)$.

In the case $w'_k = w(x, y)$, *Prim* picked (x, y) because $w(x, y) < w(u, v)$. Consider the position of $w(u, v)$ in $P(v)$, it must come after $k - 1$ since the first $k - 1$ elements are the same in $P(v)$ and $P(y)$

and (u, v) does not exist in $P(y)$. So we have $w(x, y) = w'_k > w_k \geq w(u, v)$, which contradicts with $w(x, y) < w(u, v)$.

Summarizing the two cases, we have that *DijkstraMST* and *Prim* couldn't have made two different picks as the $(n+1)^{th}$ edge, and the two construct MST in the same order.

3 Tournament scheduling

The algorithm's given in alg 1.

Algorithm 1 Tournament scheduling algorithm

```
1: input : n
2: day ← []
3: listN = [1..n]
4: function SCHEDULE(listN)
5:     n ← len(listN)
6:     if n = 1 then
7:         return listN
8:     merge(schedule(listN[1..n/2]), schedule(listN[n/2 + 1...n]))
9:     return listN
10: function MERGE(listA,listB)
11:     n ← len(listA)
12:     for n ≤ j < 2 · n do
13:         for 0 < i ≤ n do
14:             day[j].add(listA[i], listB[(i + j) mod n])
```

Correctness: this algorithm applies divide and conquer; each time we split the players into two halfs, and schedule matches inside the half. The recursion here is:

$$SchedulingResult(n) = SchedulingResult(1^{st}half) + SchedulingResult(2^{nd}half) + Merge(1^{st}half, 2^{nd}half)$$

With the base case being when each half has only one player, we schedule player $2i$ to play against player $2i + 1$ on day 1.

Assuming we have both halves of size k solved, we want to merge by scheduling the matches between the i^{th} player of the first half with the i^{th} player of the second half on day k, and scheduling the matches between the i^{th} player of the first half with the $i + j^{th}$ player of the second half on day $k + j$ until all matches are scheduled.

Time complexity: this algorithm satisfies $T(n) = 2T(\frac{n}{2}) + n^2$, and by master theorem generic form 3 (regularity condition holds), $T(n) = \theta(n^2)$.