

MthSc 816 — Assignment #2 (Kruskal's Algorithm)

Data Structures

from, to, cost: arrays for edge information	$O(3m)$
heap: array of edge indices	$O(m)$
pred, rank: node length arrays for union/find	$O(2n)$
mst: indices of edges in MST [optional]	$O(n)$

Space complexity is $O(4m + 3n)$, if G has n nodes and m edges;
or $O(4m + 2n)$, if don't store the MST but print each edge as found.

Initialize using $\text{heap}[k] = k$; $\text{pred}[j] = j$, $\text{rank}[j] = 0$

Overall Design

readgraph	$O(m)$
initialize	$O(n + m)$
makeheap	$O(m)$
while $\text{MSTedges} < n - 1$ (*)	
edge = deletemin	$O(m d \log_d m)$
root1 = findroot (from (edge))	$O(m\alpha)$
root2 = findroot (to (edge))	$O(m\alpha)$
if root1 \neq root2	
update mst , mincost , MSTedges	$O(n)$
union (root1 , root2)	$O(n)$
output MST	$O(n)$

[* Could also check that **heapsize** > 0 , if G not connected.]

Time complexity is $O(m d \log_d m)$, dominated by **deletemin** operation.

Auxiliary Routines

- minchild(x):** returns index and minimum cost of smallest child of x
if none returns minimum cost = ∞
can compute the location of the last child of x using
 $\min \{dx + 1, \text{heapsize}\}$
- siftdown(x):** restores heap order if cost at position x is increased
avoid pairwise *swaps* (instead, the vacant spot moves down)
copy edge *indices* rather than edge records
make sure it works when $\text{heapsize} = 1$

Code Issues

- all routines should be iterative — not recursive
- siftdown:** avoid special cases by having **minchild** return ∞ if no children
- makeheap:** begin siftdown at last node with a child, $\lceil (\text{heapsize} - 1)/d \rceil$
- deletemin:** returns first element of heap; copy last element of heap into the first position; reduce **heapsize** by 1; siftdown if heap nonempty

Comparison of d -Heaps

Theoretical worst-case complexity (from **deletemin**) is

$$m d \log_d m = m d (\ln m / \ln d) = (m \ln m) (d / \ln d)$$

Verify that $(2 / \ln 2) = (4 / \ln 4) > (3 / \ln 3)$, so in theory $d = 3$ should dominate *in general*. [Continuous solution at $d = e = 2.71828$]

This assumes that each siftdown takes the maximum # of steps.

Empirical Comparisons

Run for each d value, measuring total elapsed time for Kruskal (excluding reading, printing). Average CPU time is *total elapsed time*/# repetitions.

Problem	#nodes	#edges	last edge used	MST cost	CPU time
1	40	366	366	744.92	longest
2	40	375	119	635.88	shortest
3	40	475	114	634.33	near shortest

```

FUNCTION findroot(x) // trace from x to root, path halving

    while x ≠ pred[x]
        pred[x] = pred[pred[x]]
        x = pred[x]
    endwhile
    return x

FUNCTION union(root1,root2) // merge subtrees, adjust rank

    if rank[root1] ≤ rank[root2] then
        pred[root1] = root2
    else
        pred[root2] = root1
    endif

    if rank[root1] = rank[root2] then
        rank[root2] = rank[root2] + 1
    endif

FUNCTION [child, childval] = minchild(p) // position p in heap

    child = 0 // values returned if there
    childval = Infinity // are no children of p
    stop = min(d*p + 1, heapsize)

    for z = d*(p-1) + 2 : stop
        if cost[heap[z]] < childval
            childval = cost[heap[z]]
            child = z
        endif
    endfor

```

```
FUNCTION siftDown(p) // start at position p in heap

    hold = heap[p]
    val = cost[hold]
    [child, childval] = minchild(p)

    while val > childval
        heap[p] = heap[child]
        p = child
        [child, childval] = minchild(p)
    endwhile

    heap[p] = hold
```