Minimum spanning tree (MST)

Graph $G = (V, E, w)$

MST: T subgraph of G (connected) – it spans G i.e. all vertices are covered $V(T) = V(G)$

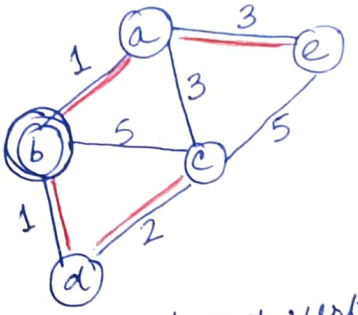
It has minimum weight.

$$w(T) = \sum_{e \in E(T)} w(e)$$

exactly $(n-1)$ edges if there are n vertices.

Prim's algorithm

root some vertex with which algorithm starts.



$T = \{b\}$;

$d[\,] = [\underset{a}{1}\ \underset{b}{0}\ \underset{c}{5}\ \underset{d}{1}\ \underset{e}{\infty}]$ // distance of each vertex form T

pick the edge (b, d)

$T = \{b, d\}$;

keep it at the old value / change for the new vertices which are not in T.

$d[\,] = [\underset{a}{1}\ \underset{b}{0}\ \underset{c}{2}\ \underset{d}{1}\ \underset{e}{\infty}]$

pick the edge (b, a)

$T = \{b, d, a\}$;

$d[\,] = [\underset{a}{1}\ \underset{b}{0}\ \underset{c}{2}\ \underset{d}{1}\ \underset{e}{3}]$

pick the edge (d, c)

~~pick the edge~~ $T = \{b, d, a, c\}$;

$d[\,] = [\underset{a}{1}\ \underset{b}{0}\ \underset{c}{2}\ \underset{d}{1}\ \underset{e}{3}]$

pick the edge (a, e)

$T = \{b, d, a, c, e\}$;

```
PRIM_MST_Seq (V, E, W, r)
{
    T = {r};
    d[r] = 0;
    for all v ∈ V\T:
        if (r, v) ∈ E : d[v] = w(r, v)
            else : d[v] = ∞ // something appropriate

    for (n-1) iterations // in each iteration 1 new edge
    {                        and 1 new vertex are added
        find u s.t. d[u] = min{d[v] | v ∈ V-T}
        T = T ∪ {u};
        for all v ∈ V-T
            d[v] = min(d[v], w(u,v));   // no edge w(u,v) = ∞
    }
}
```

n/p vertices to each processor
⇒ combine across processors in log p steps.

OpenMP parallelization

for (n-1) iterations `# pragma omp for` (u : reduction(min) d[u])

find u s.t. d[u] = min{d[v] | v ∈ V-T}

any reasonable openMP implementation will create local copy of u - reduce across processors (log p) → useful if many many processors, but in shared memory have moderate number of threads, so openMP might just use critical section.

how to get u for which d[u] holds.
find the vertex having smallest index for which d[u] = min
reduction (u : min)

for all v ∈ V-T `# pragma omp for`

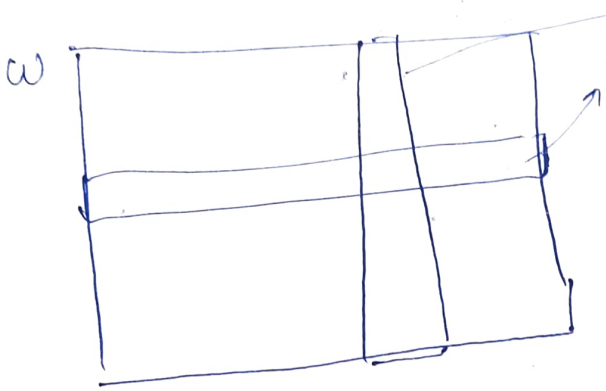
MPI implementation

reduction → MPI_Reduce, MPI_Allreduce } similar to open MP reduction

careful about which vertex is assigned to which processor?

1-D distribution — each processor n/p vertices

w matrix distributed among processors



row wise? column wise?

typically row wise keeps track of outgoing edges from a vertex.

A processor already knows $d[v]$ if we do row wise distribution, then $w[u,v]$ will be with owner of u.

* So instead of row wise, we distribute the w matrix column wise → incoming edges to v.

Time sequential time $\Theta(n^2)$

$T_p = \Theta\left(\frac{n^2}{p}\right)$ [computation]

$+ \Theta(n \log p)$ [communication for reduction each iteration is $\log p$, so for n iterations $n \log p$]

$$\text{Speedup} = \frac{\text{sequential time}}{\text{parallel time}} = \frac{\Theta(n^2)}{\Theta\left(\frac{n^2}{p}\right) + \Theta(n \log p)}$$

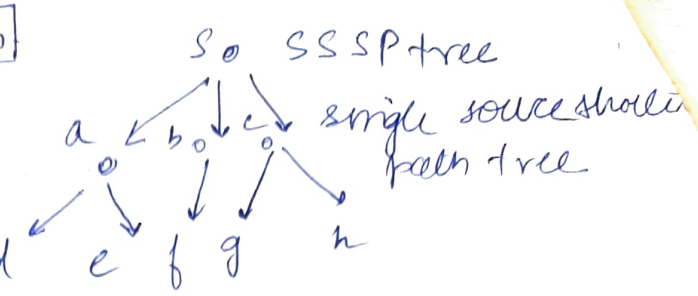
$$= \frac{1}{\Theta\left(\frac{1}{p}\right) + \Theta\left(\frac{\log p}{n}\right)}$$

$$\text{Efficiency} = \frac{\text{Speedup}}{p} = \frac{1}{\frac{p \log p}{n}}$$

Theoretical

How many processors to have ideal efficiency of 1? $p = \frac{n}{\log n}$

Single source shortest Path



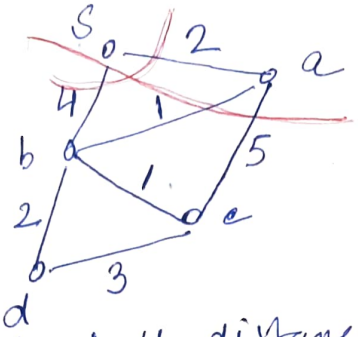
Dijkstra

$V = \{s\}$;

Shortest distance from every vertex

s	a	b	c	d
0	∞	∞	∞	∞

tentative shortest path length discovered so far



when terminates, has the shortest path distance for each vertex. can find the actual path with backtracking.

Relaxation on edge (u, v)

$$d[v] \leq d[u] + w(u, v)$$

then $d[v] = d[u] + w[u, v]$

Every edge from the vertices in V

s	a	b	c	d
0	2	4	∞	∞

Similar to Prims, I pick the minimum and include in V.

s	a	b	c	d
0	2	3	~~7~~	∞

In Prim

$d[u]$: tentative shortest edge from $T \omega V/T$

$d[u]$: tentative shortest path from $s(r)$ to u

SSSP_Seq (V, E, W, r)

{

$T = \{r\}$;

$d[r] = 0$;

for all $v \in V/T$ if $(r, v) \in E$: $d[v] = w(r, v)$

else : $d[v] = \infty$

for $(n-1)$ iteration:

find u s.t. $d[u] = \min\{d[v] \mid v \in V - T\}$

$T = T \cup \{u\}$;
for all $v \in V - T$
$d[v] = \min\{d[v], d[u] + w[u,v]\}$

Parallelization Technique is same as Prims

Speedup $\frac{p}{\Theta(i) + \Theta\left(\frac{p \log p}{n}\right)}$ $\quad p = \frac{n}{\log n}$

Sparse graph: each vertex has constant number of neighbors

$\Theta(n) \rightarrow$ for $(n-i)$ iterations
find u s.t. $d[u] = \min\{d[v] \mid v \in V - T\}$
$\Theta(n) \rightarrow$ linear search for min
$T = T \cup \{u\}$;
for all $v \in V - T$
$d[v] = \min\{d[v], du + w[u,v]\}$

should be better

$\Theta(n) \rightarrow$ for dense graph
but $\Theta(1)$ for sparse graph (as only neighbors of u can be updated)

PQ = V
for each vertex u that is selected/frozen i.e. added to SSSP tree add distance along shortest path known → removed from PQ.

① sparse graph
Use Priority Queue data structure to find minimum
$\Theta(\lg n)$ asymptotically similar

② Johnson's algorithm

$T_{seq} = \Theta(e \lg n)$ $\quad T_{parallel}$ $n\left(\left(\lg \frac{n}{p} + \log p\right) + \frac{e}{p} \lg \frac{n}{p}\right)$

How to parallelize if we need to use Priority Queue?
Each processor owns n/p vertices (Partition V into p disjoint sets $V_1, V_2 \ldots V_p$)

Each processor i maintains priority queue

Priority Queue is our vertices which are not yet finalized w.r.t. shortest path

Initialize PQi with Vi.

Maintain d as before for its own vertices.

Extracted node is settled locally and may get activated again only global mean it can be settled.

Look at all neighbors → send messages to owners of all neighbors of the extracted min vertex

for all neighbors v of u: send (owner(v), (u, v), d[u] + w[u, v])

Receiver will examine ((u, v), d[u] + w[u, v])

owner of v. checks if d[v] > d[u] + w[u, v]

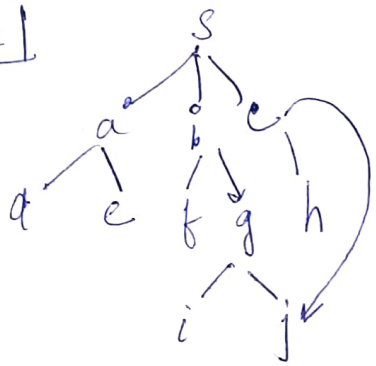
then update

* relaxation for p vertices instead of 1

iteration = diameter of the graph.

for (n − 1) iteration ← attack this.

SSSP Tree



P0	P1	P2
s	a	b, c
{s}	{ }	{ }

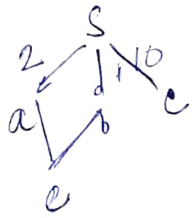
p = n

P0 → sends messages to P1, P2

Priority Queue P0{ } P1 {a} P2 {b, c}

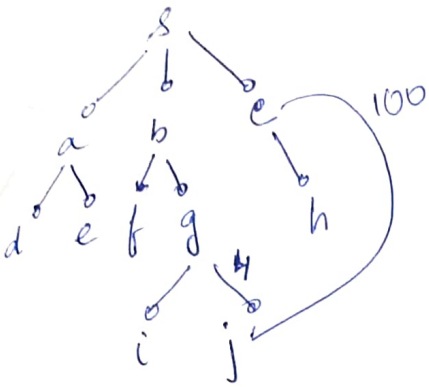
neighbors of a send msg

neighbors of b send msg



How many iterations?

depth of the tree



P_8 {b, fg} P_{10} {c, i}

itu 1	{ }	{ }
itu 2	{b} extract	{c} extract
itu 3	{f, g}	{j} extract
	↓ extract	"think" settled j
itu 4	extract	

Temporary settlement

but when P_{10} gets msg from P_8 realize j has a shorter path from s. activated again

---

Johnson_SSSP_1D (V, E, w, r)

{ if (owner(r) == myRank){
  $\bar{d}(r) = 0$;
  insert r into PQi }

d → stable value
$\bar{d}$ → tentative value we store in PQ

for all $v \in V$
  $d(v) = \infty$

while (True)
{ u = extract_min(PQ)
  $d[u] = \bar{d}[u]$
  for all neighbors $v \in Adj[u]$; ← buffer for each proc and together
    send owner(v), (u, v) ($d[u] + w[u,v]$)

Receive messages
for each received msg (x, v) $d[x] + w[x, u]$
  if u in PQ
  { if $d(x) + w[x, v] < \bar{d}(v)$
    update $\bar{d}(v)$ ← $\log \frac{n}{p}$ readjust priority queue

}

else // v not in PQ

if $(d[x] + w(x,v)) < d[v]$ → reapplication of a settled vertex (we thought it is already settled, but not).

$d[v] = d[x] + w[x,v]$

insert v into PQ ↑ readjust

in priority queue d value
outside priority queue d value

check for termination

everyone checks size of PQ

all reduce

size 0

exit