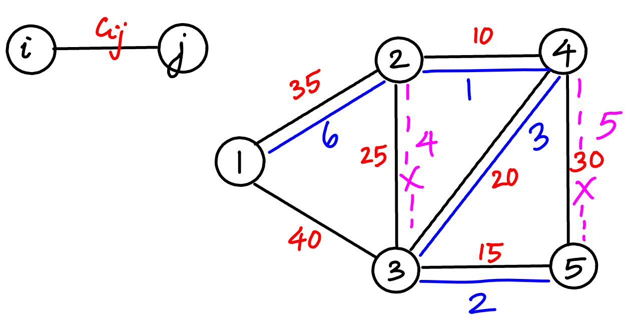
MATH 566: Lecture 26 (11/14/2024)

Today: * algos for MST - Kruskal's and Prim's algos
* Assignment and matching

Kruskal's Algorithm for Minimum Spanning Tree (MST)

Uses path optimality conditions
- builds the tree by adding one arc at a time
- uses a sorted LIST of arcs in the increasing order of c_{ij} values
- check whether adding an arc creates a cycle
  * if no, add it to tree
  * if yes, discard arc from LIST.

Example



Order in which arcs are considered indicated as $1, 2, \dots, 6$.

Arcs (2,3) and (4,5) are not added to the tree (4^{th} and 5^{th} arcs considered), as they would create cycles.

Complexity

Sorting arcs is the bottleneck step.

Sorting m arcs : $O(m \log m) = O(m \log n^2) = O(m \log n)$.

Detecting cycles when adding arcs can be done efficiently using the UNION-FIND data structure for maintaining connected components. There are two main operations/functions:

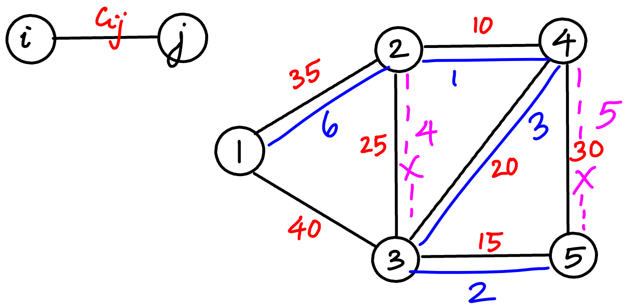
FIND(i): returns connected component containing node i
each component is represented by a single node in it

UNION(i,j) : joins components containing i and j into single component, now represented by i.

We can describe Kruskal's algorithm

```
T := ∅;   MST is empty at start
LIST of arcs (k,l) sorted according to c_kl (smallest to largest)

for each arc (k,l) ∈ LIST do
    if FIND(k) == FIND(l) then
        discard (k,l);
    else
        UNION(k,l);
        add (k,l) to tree T
    end
end
```



{1} {2} {3} {4} {5}

1. (2,4) → {②4}

2. (3,5) → {③5}

3. (3,4) → {1}, {②3,4,5}

4. (2,3) FIND(2) = FIND(3) = 2 ⇒ discard

5 (4,5) FIND(4) = FIND(5) = 2 ⇒ discard

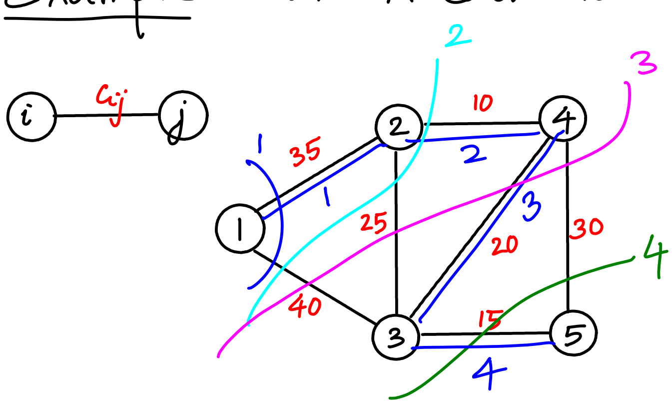
6. (1, 2) → {1,②,3,4,5}.

Prim's Algorithm

Uses cut optimality conditions

- builds a spanning tree by fanning out from a single node, adding one arc at a time
- maintains a spanning tree of the subset S of N, and adds $(i,j) \in [S,\bar{S}]$ with the smallest c_{ij} to the tree.

Example : cuts in each iteration shown: 1, 2, 3, 4



$S = [1]$ at start

$\longrightarrow [1,2] \longrightarrow [1,2,4]$

$\longrightarrow [1,2,4,3] \longrightarrow$

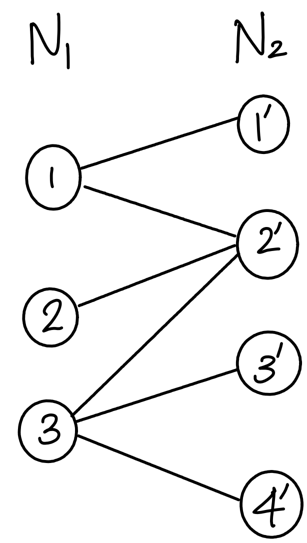
$[1,2,4,3,5]$

Sollin's algorithm combines Kruskal's and Prim's algorithms.

26.4

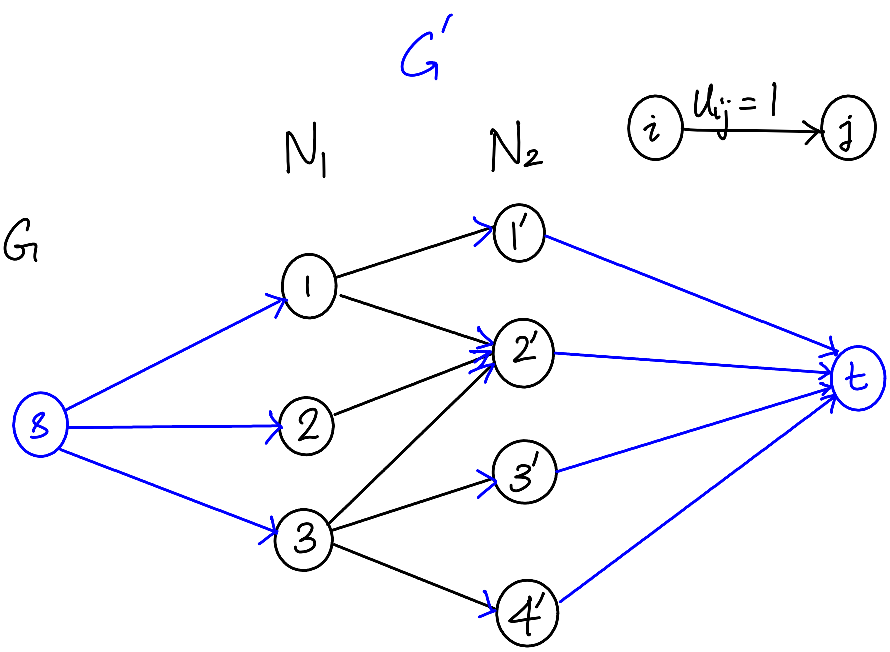
Assignments and Matching Problems (AMO Chapter 12)

1. Bipartite cardinality matching problem:
   $G=(N_1 \cup N_2, A)$ where A has undirected edges (i,j) with $i \in N_1$, $j \in N_2$. The goal is to match as many nodes in N_1 with unique nodes in N_2.



We can model this problem as a max flow problem on a **simple network** in which $u_{ij}=1 \; \forall (i,j)$, and each node i has either $\text{indegree}(i) \le 1$ or $\text{outdegree}(i) \le 1$.

Set all arc capacities as 1.



* Start with $G' = G$
* Direct all arcs $(i,j) \in G$ from i to j.
* Add nodes s, t, and arcs $(s,i)\ \forall i \in N_1$ and $(j,t)\ \forall j \in N_2$
* set all arc capacities in G' as 1.

We can show that a matching of cardinality k in G corresponds to an s-t flow in G' with value k.

($\Longrightarrow$) Given a matching of cardinality k in G, set
$x_{si} = x_{ij} = x_{jt} = 1$ in G' $\forall (i,j)$ in the matching.
Value of flow $= k$ here.

($\Longleftarrow$) Given a flow with value k in G', we use flow decomposition to obtain k path flows of the form $s-i-j-t$ with flow value 1 each. We match each such $i \in N_1$, $j \in N_2$ to define the matching. $\square$

Note that these paths will be arc disjoint (they share only the nodes s and t) since all capacities are 1.

Hence solving the cardinality bipartite matching problem on G is equivalent to solving the max flow problem on G'.

The max flow problem on simple networks can be solved in $O(m\sqrt{n})$ time (AMO Chapter 8). Hence the bipartite cardinality matching problem can be solved in $O(m\sqrt{n})$ time.

2. Bipartite Weighted Matching Problem (Assignment Problem)

$G = (N_1 \cup N_2, A)$, $|N_1| = |N_2| = n$, c_{ij}'s are costs on undirected arcs (i,j) with $i \in N_1$, $j \in N_2$. The goal is to find a perfect matching, i.e., match all nodes in pairs, of minimum total cost.

Here is the optimization model (LP) using variables x_{ij} defined to be $x_{ij} = 1$ if i and j are matched for $i \in N_1$, $j \in N_2$, and $x_{ij} = 0$ otherwise.

linear program

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{(i,j)\in A} x_{ij} = \overset{b(i)=1}{1} \quad \forall i \in N_1 \rightarrow \text{supply nodes}$$

$$-\left(\sum_{(i,j)\in A} x_{ij} = 1\right) \quad \forall j \in N_2 \rightarrow \text{demand nodes with } b(j) = -1$$

$$0 \le x_{ij} \underbrace{\le 1}_{} \quad \forall (i,j) \in A$$

upper bound is implied by the perfect matching constraints

Special case of the min-cost flow problem with the nodes in N_1 being supply nodes (all with unit supplies) and nodes in N_2 are demand nodes (all with unit demands).

Successive Shortest Path Algorithm (for assignment problem)

Recall $(\bar{x}, \bar{\pi})$ are optimal for MCF iff $c_{ij}^{\bar{\pi}} \geq 0 \;\forall (i,j) \in G(\bar{x})$.

Here, augmenting 1 unit corresponds to assigning one additional node in N_1. We augment 1 unit in each iteration.

If $S(n,m,C)$ is the complexity of solving the SP instances, the assignment problem could be solved in $O(n\,S(n,m,C))$ time.

A Relaxation Algorithm

Allow nodes in N_2 to be over- or under-assigned. Here, under-assigned means not assigned at all. We then pick node $k \in N_2$ that is over-assigned, find SP from k to all nodes in $G(\bar{x})$ with $c_{ij}^{\bar{\pi}}$ as costs. Augment 1 unit along shortest path from k to some $l \in N_2$ that is underassigned.