CS 6381: Lecture #17 10·3·2020

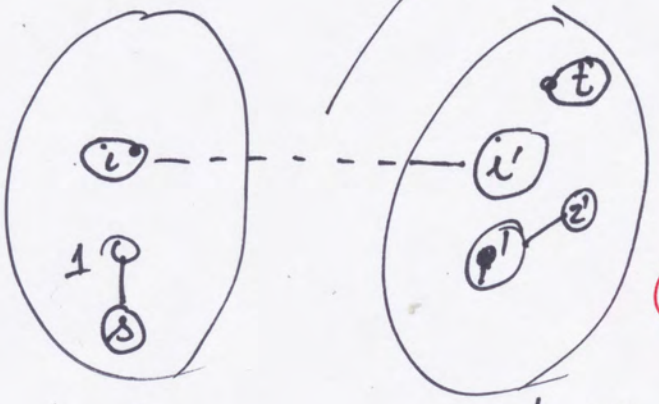
Applications of Matching & Extensions of Matching

Shortest Paths in undirected graphs without <0 cycles

Input: $G = [V; E]$ edge weights such that $\nexists$ neg. cycles. ; s: origin, t: destination

Odd/Even SP

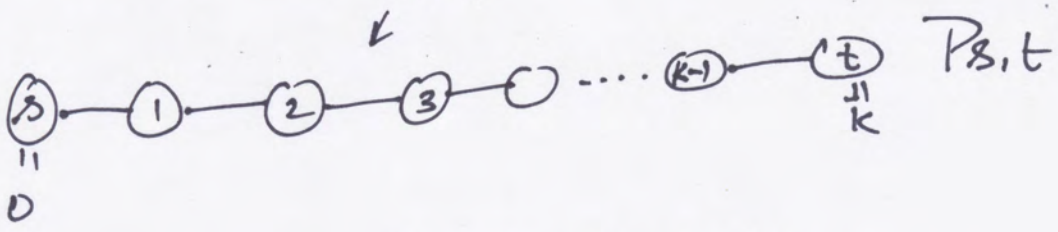
(1)



$w(i, i') = 0 \ \forall \ i \neq s, t$

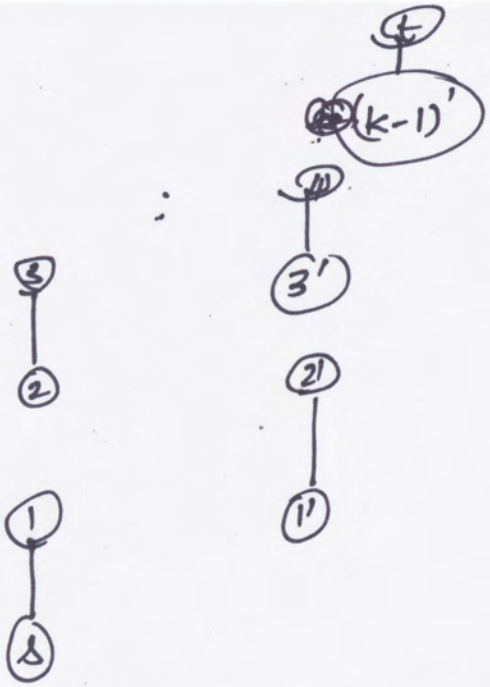
(This works since no neg. cycles!)

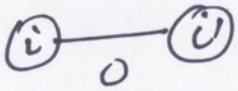
Solve PMP problem. If there is a s-t path in G, let $P_{s,t}$ be a SP.



Consider edge $(s-1)^*$, $(1', 2')$, $(2, 3) \cdots$

See next page Such Paths have even # of edges



+ nodes i that are not in $P_{s,t}$: 

Since there are no negative cycles, this is the best perfect matching

The transformation works in both directions

PM → Path; + 0 wt edges

Path + 0 wt edges → PM.

With same cost.

This gives us best path with even # of edges

To get odd # edges best path, keep (s) and (t) in G and remove both in G'.

Same method works. One of these is best overall.

This process also works to detect neg. cycles in Weighted Undir. graphs. in Strongly Poly. time.

Remark: Actually, any SP alg. for undir. graphs (detecting -ve cycles) also Can Solve PM problem. If PM is not min cost, there exists an alt. cycle which is negative with respect to that PM. and hence can be used to improve the PM.

Ref ? PhD Student from Michigan

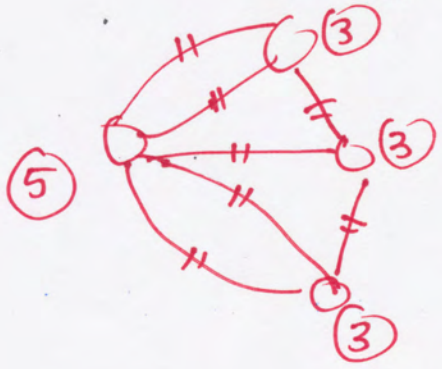
(Chinese) Postman Problem & Eulerian Graphs

L. Euler: Can we check if all edges of an undirected graph be traversed so that each edge occurs only once in a cyclic walk (possibly most nodes are revisited) ?

(Seven Bridges of Königsberg)

Answer: G must be connected and degrees (# of edges) at each node is even

Bridges of Königsberg



degree at each node is odd

So Not possible

G is NON. Eulerian

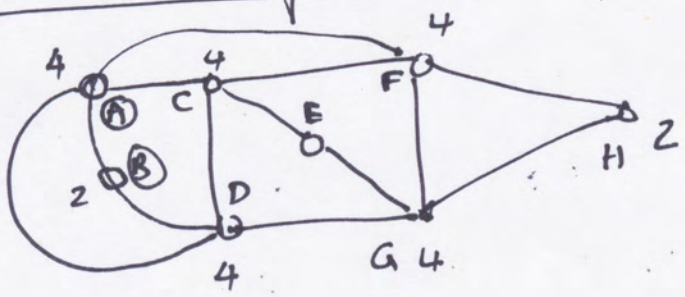
Necessity is obvious. Since each time you come into a node you must get out w/o repeating the same edge.

Sufficiency:

Assume deg (i) = even for all i

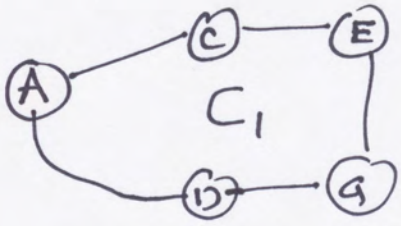
G is Connected.

Cycle Decomposition

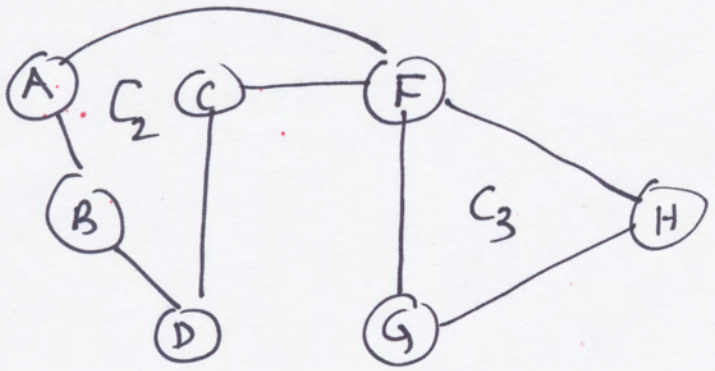


Start at any node with deg ≥ 0

go to some other

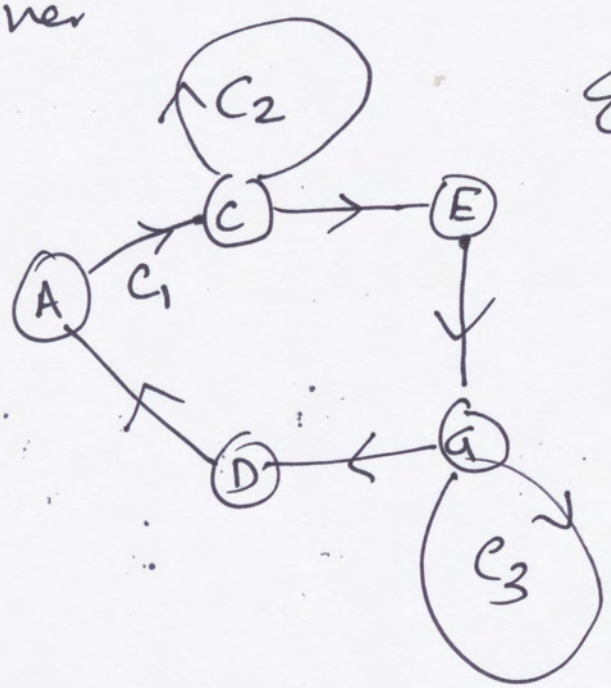


Remaining graph



$G = C_1 \cup C_2 \cup C_3$

Now traverse these cycles in a Depth-First-manner



Eulerian Traversal

That this can be done in general should be clear. Hence these conditions are also sufficient.

What if G is connected but degrees are not even? We will have to traverse some edges more than once for an Eulerian traversal.

Here is where Postman problem occurs. We want to minimize total distance travelled given edge weights; traverse each edge at least once (in either direction),

More questions: 1. Do we have to traverse more than twice on any edge?

2. Is this problem polynomially solvable?

3. What is the corresponding problem for directed graphs?

We will answer all of these. First we restrict ourselves to (the harder problem) on undir. graphs. First a few preliminary results.

Lemma For any undir. graph, # of nodes with odd-degree is even.

Proof on the next page.

Since each edge is counted once in computing degree at each of its ends,

$$\sum_{i \in V} \deg(i) = 2|E| = \text{even}$$

$$\| $$

$$\sum_{i:\text{odd}} \deg(i) + \underbrace{\sum_{i:\text{even}} \deg(i)}_{\text{even}}$$

$$\therefore \quad \sum_{i:\text{odd}} \deg(i) = \text{even}$$

$\Rightarrow$ # of odd degree nodes is even.

For the postman problem if G is connected (which we assume from now on), and all nodes have even degree, by cycle decomp. we have shown that each edge needs to be traversed <u>exactly</u> once to get Eulerian tour (we assume +ve edge weights here) We discuss the other case next.

Formulation:

Lemma: No edge is traversed more than 2 times in any optimal solution.

Pf: If e is traversed 3 or more times, remove 2 of them. If degrees are even before, they are also so after. Moreover, graph is still connected.

So some edges are traversed exactly once and others are traversed exactly twice. Consider the subgraph G_2 containing only those edges that are traversed twice in some optimal solution. $G_2 = [V; E_2]$

E_2: (edges that are traversed twice)

Lemma: G_2 can not have cycles.

Pf: If there is a cycle $C \in G_2$, reducing the number of times these edges are traversed by one, preserves even degree and connectivity. Hence the result.

Hence G_2 is a forest. (a bunch of trees)

$$\deg_{G_1}(i) + \deg_{G_2}(i) = \text{even} \quad \forall i \in V$$

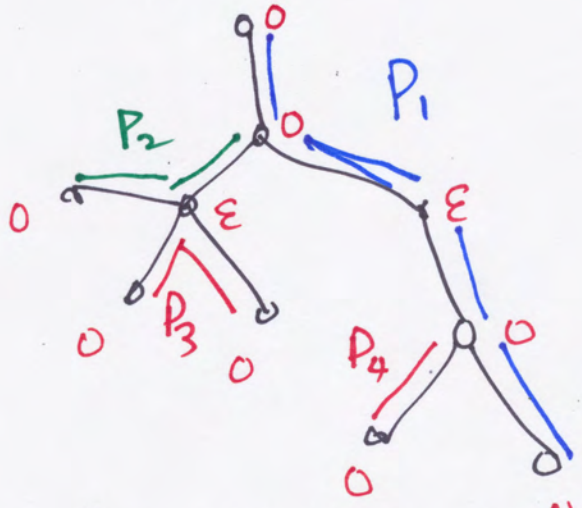
$\therefore$ If $\deg_{G_1}(i)$ is odd (even) then so is

$\deg_{G_2}(i)$ odd (even)

Certainly, leaf nodes in G_2 have deg = 1 (odd).

Lemma: G_2 ~~be~~ can be decomposed ~~to~~ into edge disjoint paths between odd degree nodes.

Example: Select any ~~tree in~~ component of G_2



Hence "replication" graph can be decomposed into edge disjoint Paths between odd degree nodes

might as well use SP between these

Algorithm

Step 1 Find SP between all pairs of odd degree nodes of G. [use FW Algorithm; all edge weights are +ve].

↗ complete graph

Step 2 Create a new graph $H = [U; F]$

U: odd degree nodes of G.

F: an edge between each pair of nodes in U with weight equal to length of a SP between those nodes in G.

Recall |U|: even

Step 3 : Solve PMP (P3) in H with above weights.

Step 4 : Replicate edges in the paths in Step 3 in G. G is now connected & Eulerian.

Step 5 : Traverse G with replication (this is Union of edge disjoint cycles!)

2-machine UET Scheduling with precedence Constraints

Problem: There are 2 identical machines
n jobs each requiring 1 unit of time
(on ~~or~~ either machine: only one)
Jobs have a partial ordering
reflecting precedence Constraints.
Want to minimize makespan →
time to finish all jobs.
Discover the schedule.

In the schedule, if each machine has a job in a certain "time slot", then these must be unrelated: neither precedes the other (either directly or by implication). Such jobs are called independent. The more such time slots, less the makespan. We use this to convert this to a matching problem and show how to recover the actual schedule.

See next page

Construct a graph $G = [V; E]$

V: one node per job.

$(i, j) \in E$ if and only if i and j are unrelated

(i.e can be done at the same time slot)

Algorithm:

Find a maximum cardinality matching M in G. Construct the schedule given M.

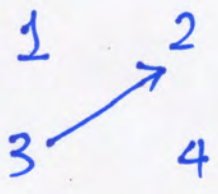
Corresponding to any feasible schedule with k pairs of jobs (in the same time slot), since jobs in a pair are independent, there is a matching in G of size k. To show the converse, is the main Contribution of FKN. This will justify the Statement that this is an application of matching problem. This we do on the next page.

If there is a singleton (node that is free in Matching) which has no predecessors, schedule it in the first time slot and use induction for the rest.

So we now assume no such singleton is present.

But there are jobs without predecessors — however these will be paired with other jobs in this case. Status of the other job with which job without predecessor is paired might vary. If both jobs in a pair are without any predecessors, we can schedule them first and use induction for the rest.

So we suppose this is not the case. For the sake of clarity, job 1 is without predecessors but it is paired with 2 that has predecessors.



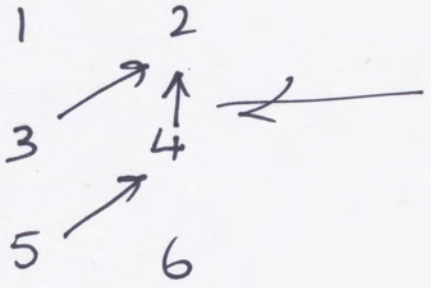
Look for a predecessor of 2 without predecessors (it exists!) Call it job 3.

3 is paired (else it goes in the first slot) say with 4. If 2 and 4 are independent, exchange

1 3 ← Schedule in first slot

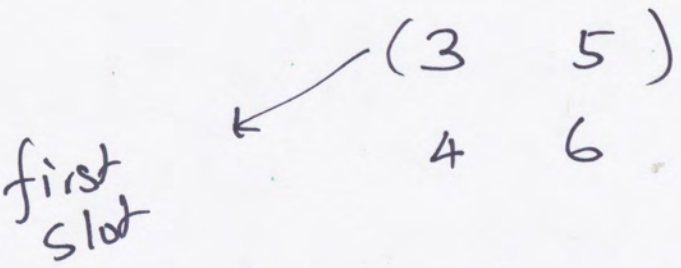
2 4

So assume not.



else (3,4) can not be a pair.
Let 5 be a predecessor of 4 (without pred.)

If 5 is a singleton, schedule in first time slot.
So it is paired with 6. If 4 and 6 are independent



So this is not the case so ~~4~~ 6 → 4

Keep repeating this and since # of jobs is finite we must have the possibility of an exchange and we schedule a pair in the first slot & use induction for the rest.

This completes proof that this is a valid application of Matching.