COS 423 Lecture 19
Graph Matching

Given an undirected graph, a *matching* is a set of edges, no two sharing a vertex. A vertex is *matched* if it has an end in the matching, *free* if not. A matching is *perfect* if all vertices are matched.

Goal: In a given graph, find a matching containing as many edges as possible: a *maximum-size* matching

Special case: Find a perfect matching (or verify that there is none)

Generalization to *weighted matching*: each edge has a weight

Goal: Find a matching of maximum total weight.

Variants: Find a perfect matching of maximum (or minimum) total weight; among maximum-size matchings, find one of maximum (or minimum) total weight

Important special case: bipartite graphs

A graph is *bipartite* if its vertices can be colored with two colors such that each edge has ends of different colors

Four versions of matching

unweighted, bipartite

unweighted, general

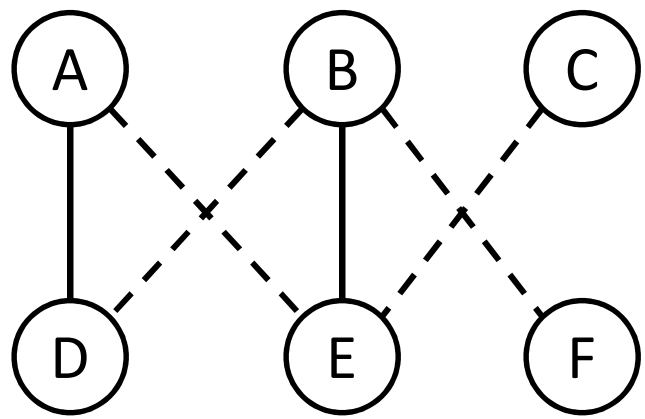
weighted, bipartite: *assignment problem*

weighted, general

A bipartite graph

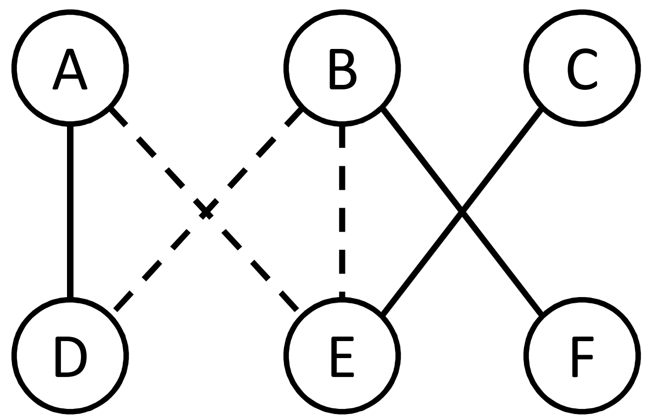
Solid edges are a matching

(maxim**al** but not maxim**um**)



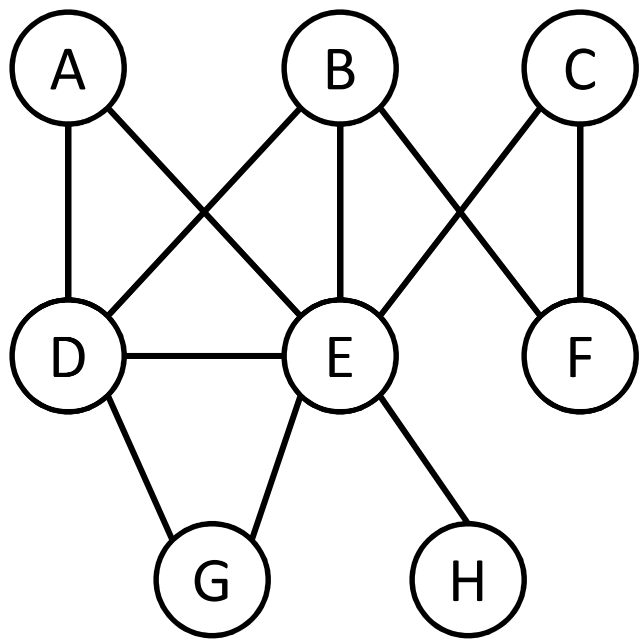
A *maximal* matching is one to which no additional edge can be added

Another matching, perfect hence maximum

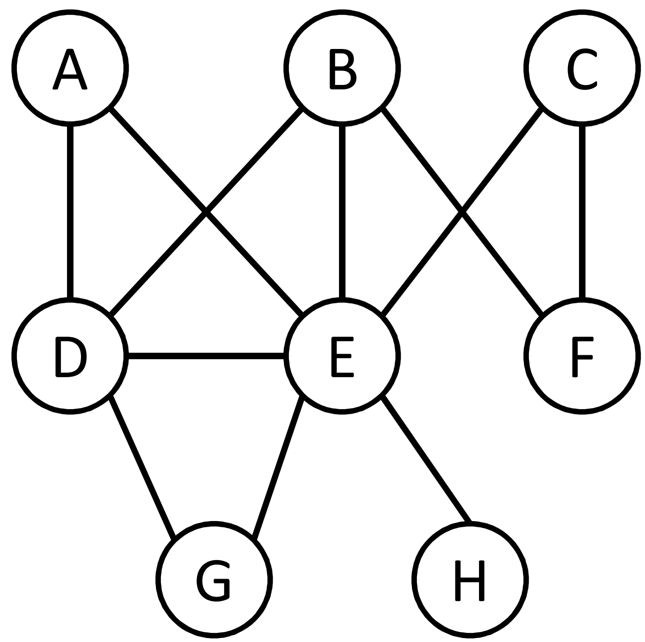


A nonbipartite graph

Does this graph have a perfect matching?



No: Each of A, G, H must be matched to D or E



Efficient matching algorithm?

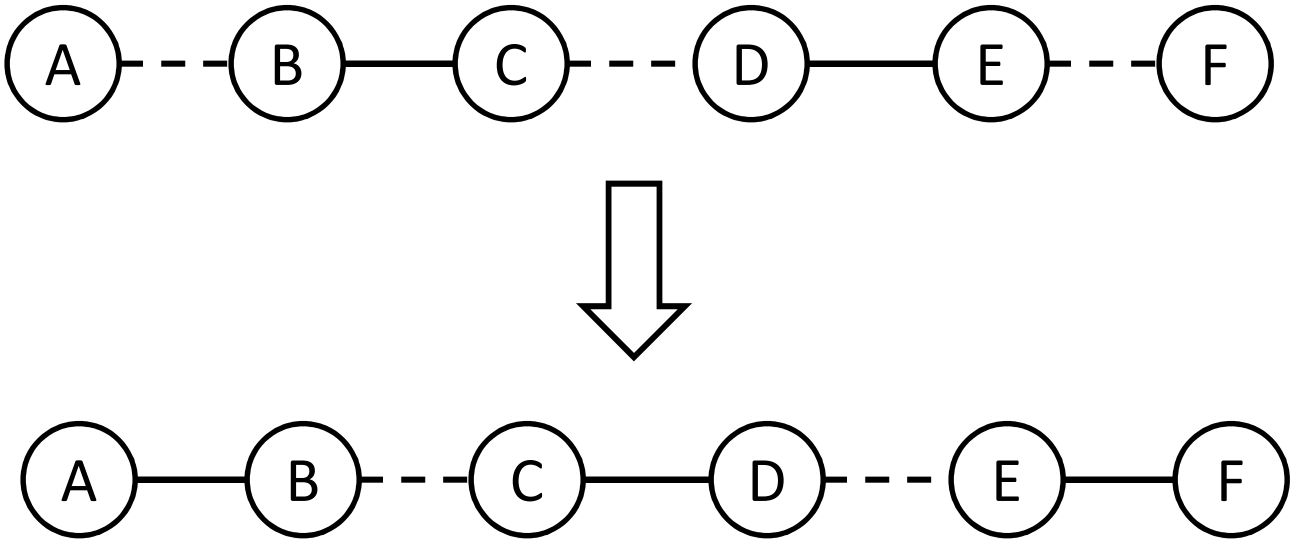
Iterative improvement: Start with any matching. Find a way to improve it by making local changes. Repeat until no improvement is possible. Hope: Any local maximum is a global maximum

Alternating path: a path whose edges are alternately in and out of the matching

Augmenting path: an alternating path between two free vertices

Augmentation: given an augmenting path, change its unmatched edges to matched and vice-versa, increasing the size of the matching by one

A, F free



A, F matched

Augmenting path algorithm

Start with the empty matching. While there is an augmenting path, do an augmentation.

Theorem: A matching has maximum size iff there is no augmenting path

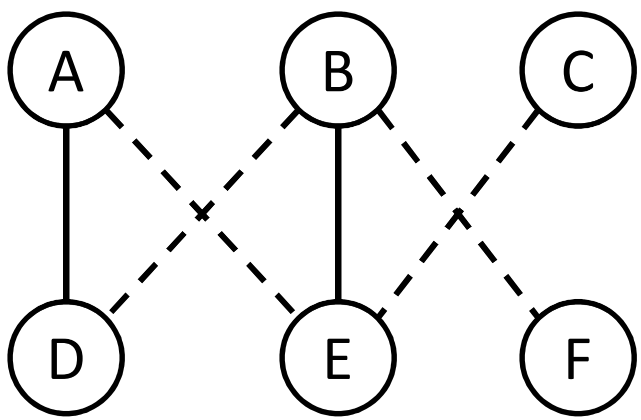
Proof: to follow

How to find augmenting paths?

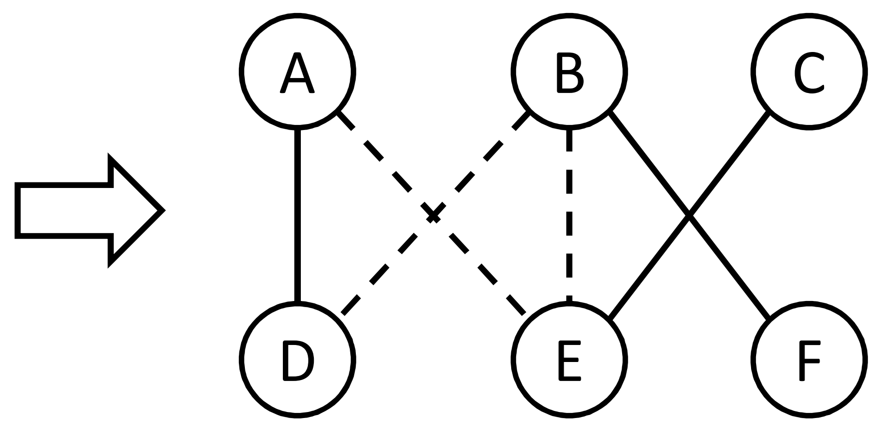
How to choose augmenting paths?

augmenting path

C, E, B, F



after augmentation



Matching Theorem: Let M be any matching, let M' be a maximum-size matching, and let $k = |M'| - |M|$. Then M has k vertex-disjoint augmenting paths

Proof: Let $M' \oplus M$ be the *symmetric difference* of M' and M, the set of edges in M' or M but not both. Each vertex is incident to at most two edges in $M' \oplus M$. The connected components of the subgraph induced by the edges in $M' \oplus M$ are thus simple paths and simple cycles.

Proof (cont.): On each such path or cycle, edges of M' and M alternate. Each cycle contains the same number of edges in M' as in M. Each path contains the same number of edges in M' as in M to within one. A path that contains one more edge of M' than M is an augmenting path for M. In $M' \oplus M$ there are exactly k more edges in M' than edges in M. Thus the subgraph induced by the edges in $M' \oplus M$ contains k vertex-disjoint augmenting paths for M (and no augmenting paths for M').

Corollary: If M is a matching whose size is k less than maximum, then M has an augmenting path of at most n/k vertices.

Both the theorem and its corollary are true for *all* graphs, not just bipartite ones

Algorithm for bipartite graphs

Let X, Y be the bipartition of the vertices.

Begin with the empty matching.

Direct all edges from X to Y.

while $\exists$free vertex x in X **do**

{search from x until reaching a free vertex in Y or finishing search;

if free vertex in Y reached **then** augment and reverse directions of all arcs on the augmenting path

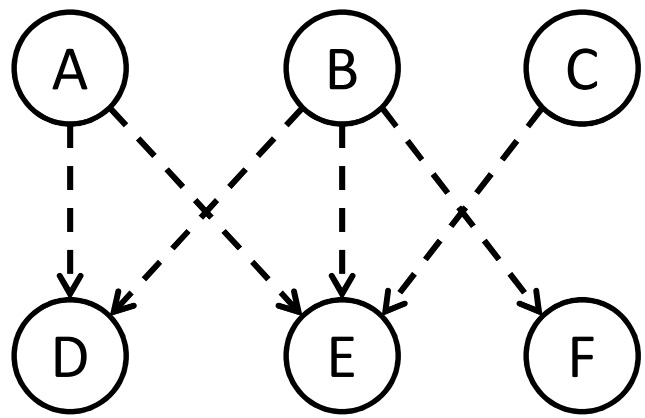
else delete all visited vertices}

Proof of correctness: Exercise.

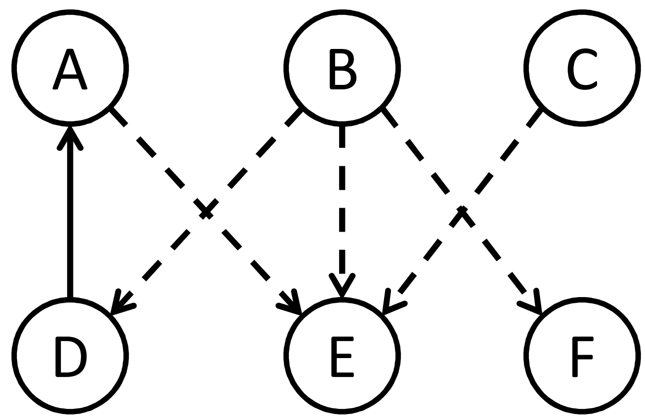
Must show that deleted vertices can never be on an augmenting path

Can also search from all free vertices in X simultaneously (stay tuned)

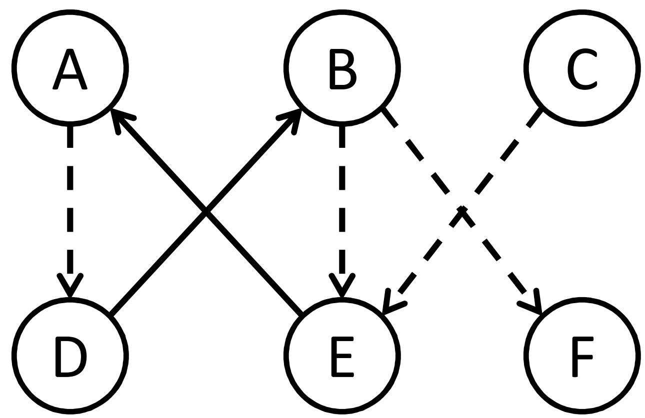
Search from A, find A, D, augment



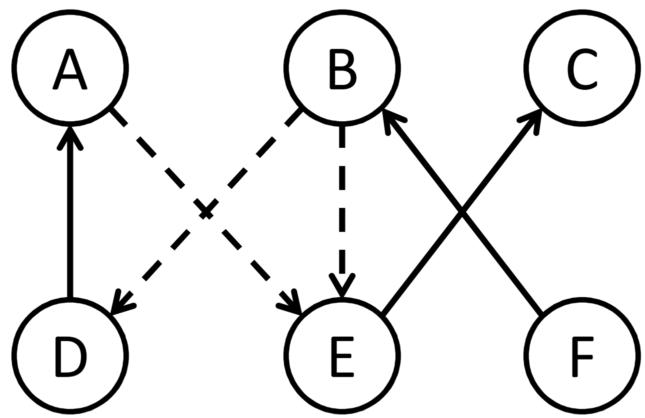
Search from B, find B, D, A, E, augment



Search from C, find C, E, A, D, B, F, augment



Search from C; find C, E, A, D, B, F; augment



O(m) time per search, O(nm) total time

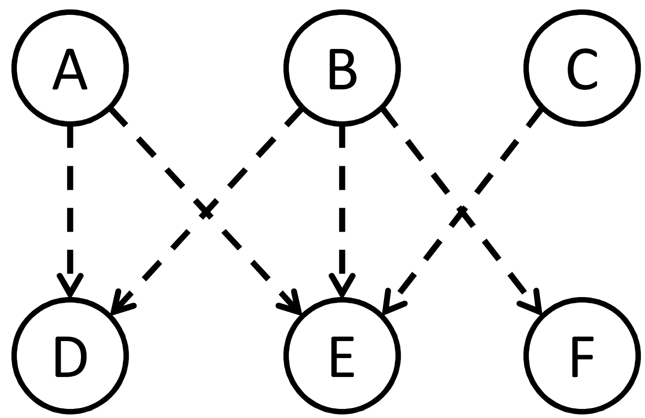
Faster: Hopcroft-Karp algorithm

Do a BFS from all free vertices in *X* concurrently (add all to initial queue) to form a *layered subgraph L* containing *all* shortest augmenting paths: truncate the search at the level of first free vertex in *Y* reached. Find vertex-disjoint augmenting paths in *L* by DFS, at most one (tree) path per start vertex. Augment along all paths found (one *phase*).

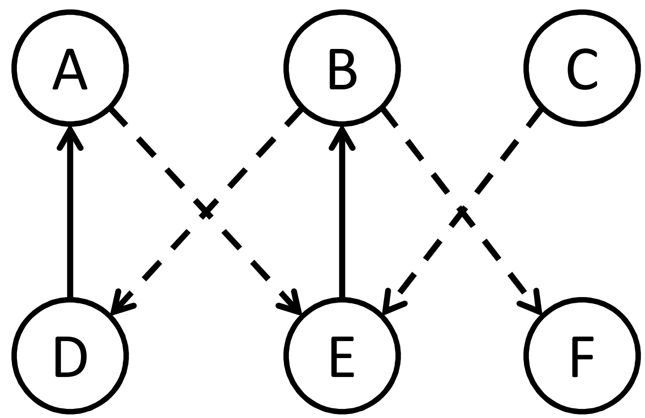
Repeat until BFS finds no augmenting path.

BFS from A, B, C; *L* is entire graph

DFS finds paths A, D; B, E; augment

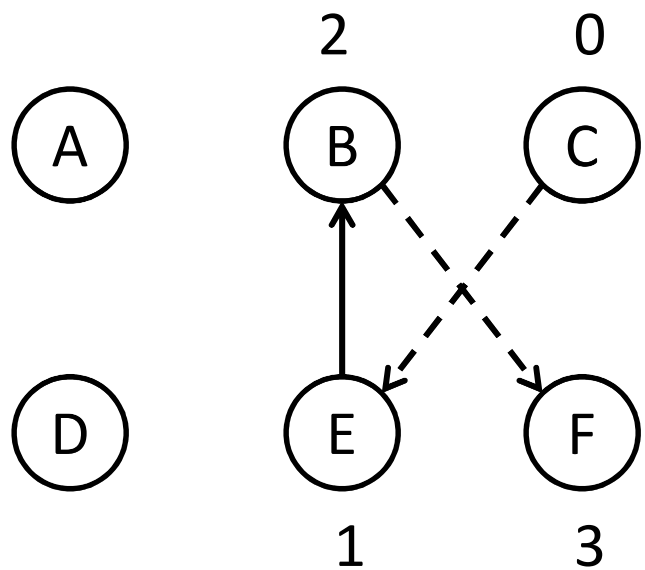


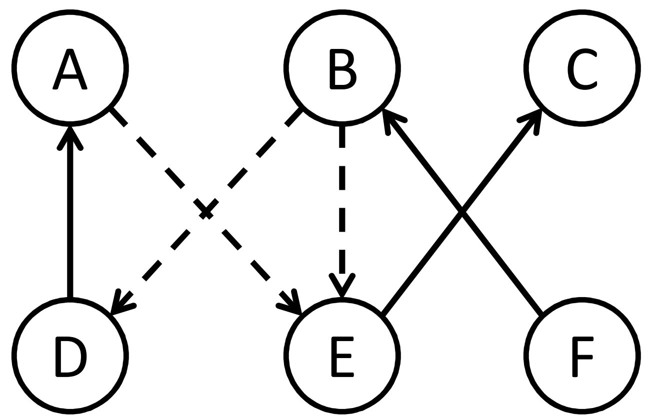
After first phase, matching is *maximal*: no edge can be added



BFS from C to form *L*

DFS from C finds C, E, B, F; augment




A
B
C
D
E
F

O(m) time per phase, $\le (2n)^{½} + 1$ phases

$\rightarrow$ O($n^{½}m$) total time

Proof: Each phase takes O(m) time. Consider a given phase, and let $d(v)$ be the minimum number of edges on an alternating path from a free vertex in X to v, just before the phase. Each arc (v, w) satisfies $d(w) \le d(v) + 1$, with equality if the arc is in L. Let k be the fewest number of arcs on an augmenting path.

Proof (cont.): Once L is constructed, it contains every free vertex in Y reachable from a free vertex in X by an augmenting path of k vertices. Each arc (v, w) on an augmenting path found by the algorithm has $d(w) = d(v) + 1$. The augmentation reverses the arc, so that $d(v) = d(w) - 1$. Suppose that, after the augmentations, there were an augmenting path of k or fewer edges. Then all such edges would have to be in L when L was first built, and the path would be found by the DFS.

Proof (cont.): We conclude that after the phase, any augmenting path contains at least $k + 2$ edges. (The number of edges on an augmenting path is odd.)

Each phase except the last one does at least one augmentation. After j phases, the length of the shortest augmenting path is at least $1 + 2j$. By the corollary to the Matching Theorem, the current matching is within $n/(1 + 2j)$ of maximum size, so there can be at most $n/(1 + 2j) + 1$ additional phases.

Proof (cont.): Thus the total number of phases is at most $j + n/(1 + 2j) + 1$. Choosing $j = (n/2)^{½}$, we conclude that the number of phases is at most $(2n)^{½} + 1$.

No faster method is known, although with this method the total length of all augmenting paths is $O(n \lg n)$: Could there be an $O(n^2 \lg n)$-time algorithm?