v0.1: Seth Gilbert
v0.8: Steven Halim

# CS4234
# Optimiz(s)ation Algorithms

## L8 – (Weighted) Max-Cardinality -(Bipartite)-Matching

https://visualgo.net/en/matching

# CS3233/CS4234
# Dual Slides ☺

icpc regionals 2018
advancing the art and the sport of competitive programming
icpc.foundation

Dr. Steven Halim

Social Development Network

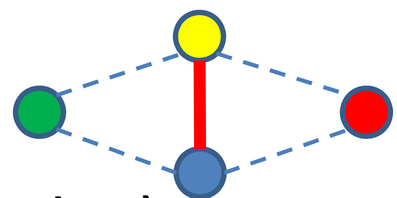
# Roadmap (for CS4234)

## Graph Matching

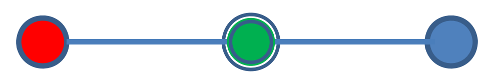
- Overview
- Unweighted MCBM: Max Flow review, Augmenting Path, Hopcroft Karp's (nice *theoretical* result), Augmenting Path++
    - A few relevant applications will be discussed in tutorial ☺
- Weighted MCBM: (Min/Max) Cost (Max) Flow, Hungarian (**overview**)
- Unweighted MCM: Edmonds's Matching (**overview**)
- Weighted MCM: DP with Bitmask **(small graph only, review)**
    - Sorry, still unable to make it work for Christofides's 1.5-Approximation algorithm for M-R/NR-TSP as of year 2018

# Graph Matching

A matching (marriage) in a graph **G = (V, E)** (real life) is a subset **M** of **E** edges in **G** (special relationships) such that no two of which meet at a common vertex (no affair)

Thus a.  and b. 

are matchings (red thick edge),

But trying to match both edges in c. 

is invalid since there is an overlapping vertex

# `Max-Cardinality-Matching` (MCM)

Usually, the problem asked in graph matching is the size (cardinality) of a <u>maximum</u> (not maximal) matching

A maximum matching is a matching that contains the largest possible number of edges

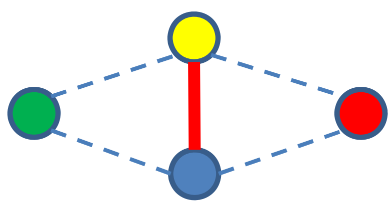
A possible variant: Perfect Matching

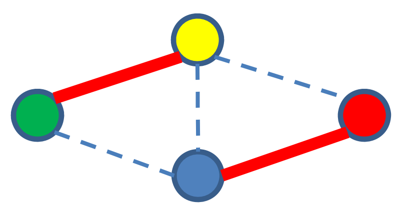
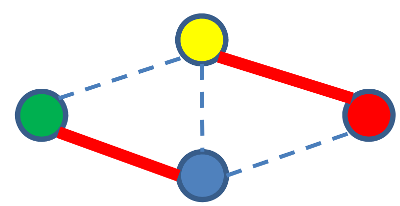
- MCM with no vertex left unmatched
  - PS: This is *impossible* on graph with odd number of vertices

# Examples

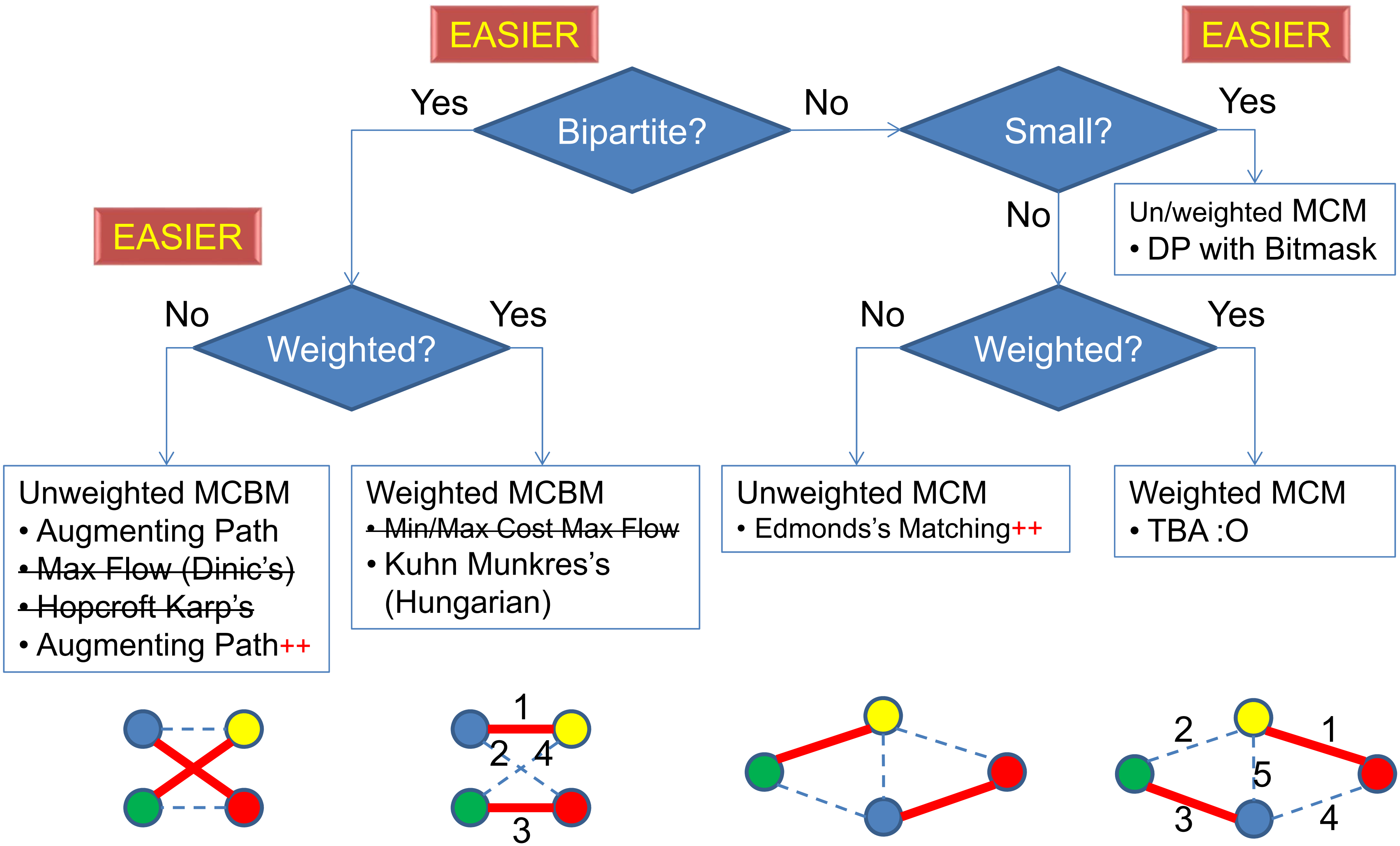
 is a max(imum) matching (0 matching)
(no edge to be matched)

 is also a max(imum) matching (1 matching)
(no other edge to be matched)

But  is not a max(imum) matching
(it is a max(**imal**) matching btw)

as we can change it to  or 
(2 matchings)

# Types of Graph Matching



Solutions:

Max Flow (just use Dinic's, one of the best for bipartite matching graphs)

Augmenting Path Algorithm

Hopcroft Karp's Algorithm ~= Dinic's (for theoretical interest only)

Augmenting Path Algorithm++ (the ++ is very important :O)

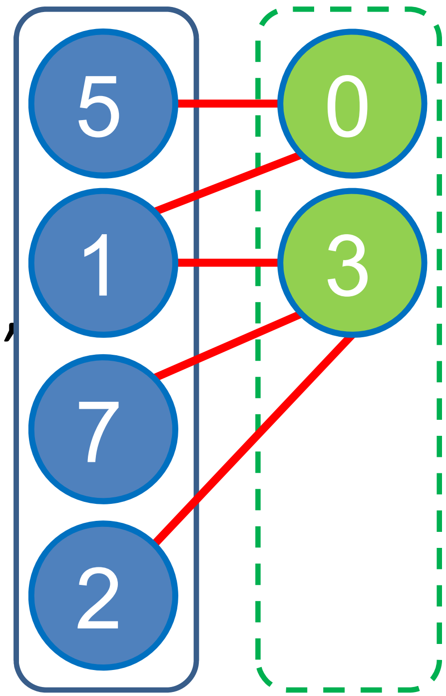
# UNWEIGHTED MCBM

# Max-Cardinality-Bipartite-Matching (MCBM)

A **Bipartite** Graph is a graph whose vertices can be divided into two disjoint sets **X** and **Y** such that every edge can only connect a vertex in **X** to one in **Y**

Matching in this kind of graph is **a lot easier** than matching in general graph

So if we are given a graph matching problem, **<u>our first check</u>** is to see whether the given input graph is bipartite…



See https://visualgo.net/en/maxflow, select modeling, Bipartite Matching, all ones

Time Complexity of such modeling:
Depends on the chosen Max Flow algorithm, **but much faster** than general case as the capacities are **all ones (unit weights)** and the graph **is bipartite**, e.g. O($\mathbf{m^2}$) for basic FF, O(**sqrt(n)*m**) for Dinic's

Finding MCBM by reducing this problem into

# MAX FLOW
# CP3.17B SECTION 4.6, 4.7 & 4.8.4

# CP3.17b Ex 4.8.4.1*: Why the graph has to be directed?

- Try creating a counter example

# When To Use Max Flow Solution?

Only if we are solving Bipartite Matching
**with Capacity** (the “Assignment Problem”)

- e.g. UVa 00259, CP3.17b Section 4.6.5

See https://visualgo.net/en/maxflow, select modeling, Bipartite Matching, random

- This variant will be **much slower** to solve if reduced to MCBM, imagine if the typical capacities of the edges are big, reducing the problem to MCBM will lead to a gigantic bipartite graph…

But if we are solving pure MCBM (capacities are all 1), just use the next algorithm (easier to code and faster)

Finding MCBM via

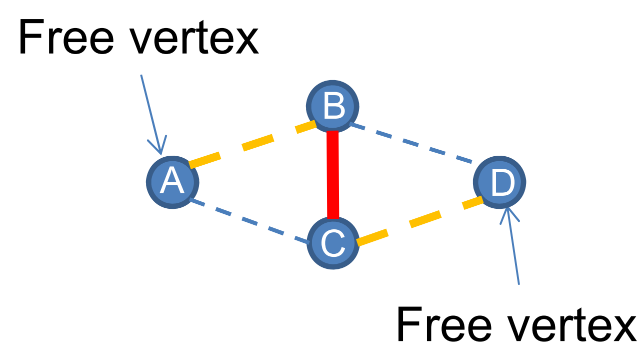
# AUGMENTING PATH ALGORITHM
# CP3.17B SECTION 4.8.4++

CS3233/CS4234
Steven Halim, SoC, NUS

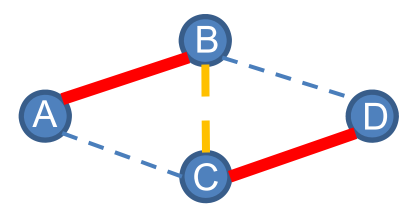
PS: Slightly different from similar term in Network Flow lecture

# Augmenting Path

In this graph, the path colored **orange(unmatched)-red(matched)-orange**: A-B-C-D is an augmenting path




We can flip the edge status to **red-orange-red**: A-B-C-D and the number of edges in the matching set <u>increases by 1</u>



# Augmenting Path Algorithm

Lemma (Claude Berge 1957):

Detailed proof omitted, you can search for external resources if you are interested

A matching **M** in **G** is maximum
iff there is no more augmenting path in **G**

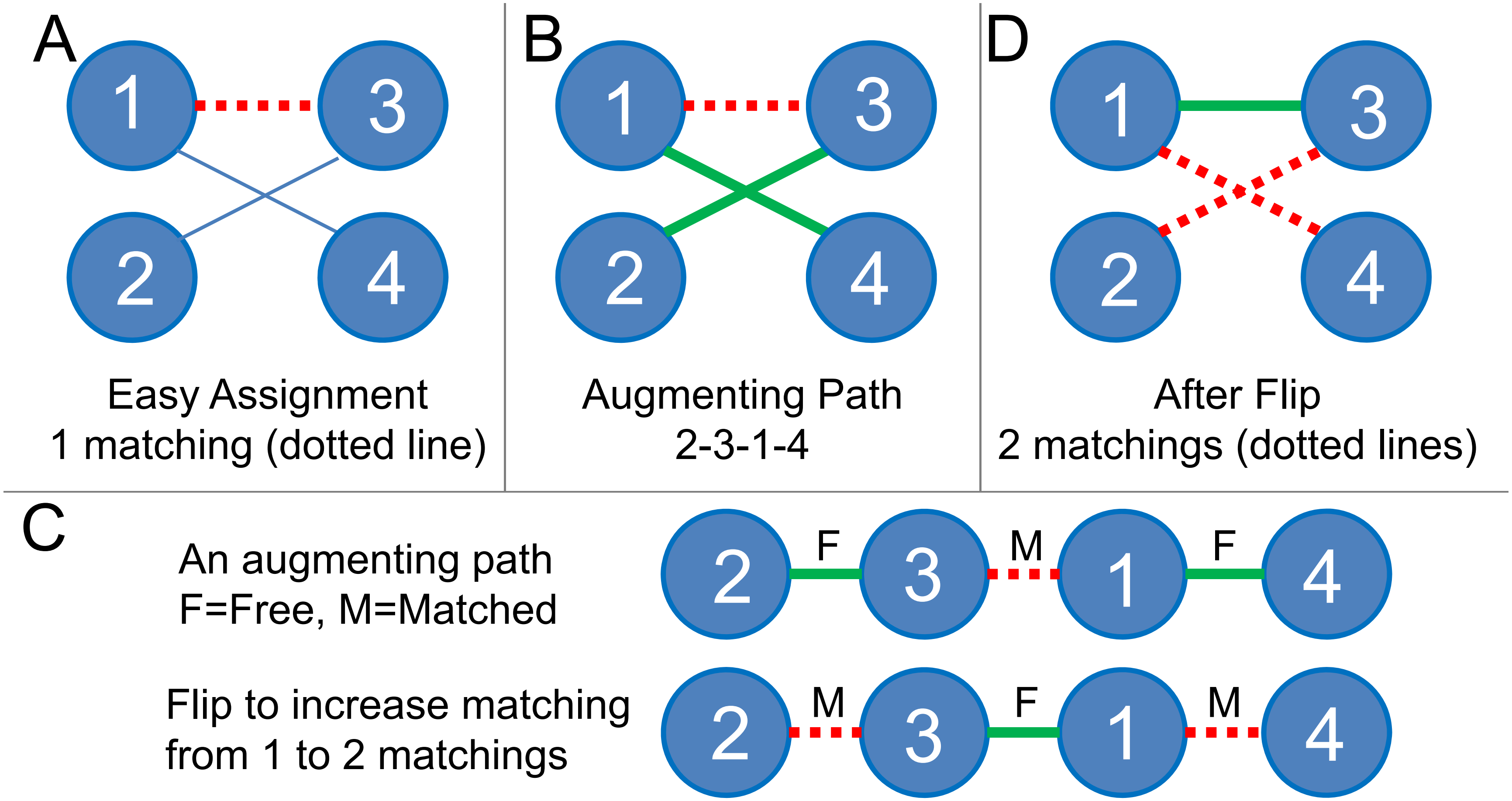
G does not necessarily be a Bipartite Graph in the lemma

Augmenting Path Algorithm is a simple $O(\mathbf{V}*(\cancel{\mathbf{V}+}\mathbf{E}))$ ~= $O(\mathbf{VE})$ implementation of that lemma

- Find and then eliminate augmenting paths in **G**
  - PS: Later we will see an $O(\mathbf{kE})$ implementation of this thing, $\mathbf{k} << \mathbf{V}$

https://visualgo.net/en/matching, try it there ☺

# Augmenting Path Algorithm



Finding MCBM via

# HOPCROFT KARP'S ALGORITHM (SECTION 9.13)

CS3233/CS4234
Steven Halim, SoC, NUS

https://visualgo.net/en/matching, load (almost) complete Bipartite Graph, and see how slow the standard Augmenting Path algorithm can be…

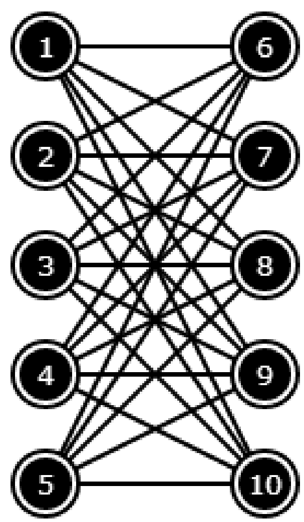
# An Extreme Test Case…

A Complete Bipartite Graph $K_{\mathbf{N},\mathbf{M}}$, **V=N+M** & **E = N*M**

Augmenting Path algorithm → O((**N**+**M**)*(**N***M**))

- If **M** = **N**, we have an O($\mathbf{N}^3$) solution, only OK for **N** ≤ 200

Example with **N** = **M** = 5

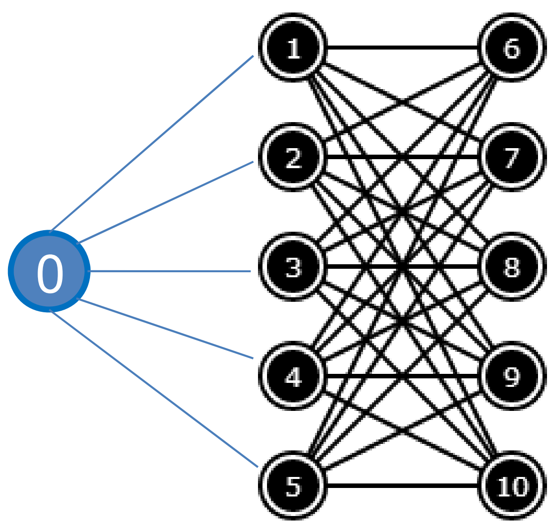


https://visualgo.net/en/matching, reviving HK animation by Week 09 or 10??

# Hopcroft Karp's Algorithm (1973)

Key Idea (very similar to *Dinic's algorithm**):

- Use shortest augmenting paths from all free vertices (BFS)
- Run similar algorithm as the Augmenting Path Algorithm earlier (DFS), but now using this BFS information,
- Can have ≥1 matching(s) per iteration



# Hopcroft Karp's Algorithm (1973)

That is, up to √V iterations only

Hopcroft Karp's runs in O(√**V** * **E**), proof omitted

- For the extreme test case in previous slide,
  this is O(sqrt(**N**+**M**)***N*****M**)
- With **M** = **N**, this is about O(**N**$^{5/2}$), OK for **N** ≤ 600
  - PS: Dinic's algorithm on Bipartite Matching graph also runs in O(N$^{5/2}$)

CP3.17b Ex 4.8.4.3*?: Is this algorithm **must be learned** in order to do well in practice$_{\text{/programming contest}}$?

- CP Answer: NOT needed…
  - You can make the standard augmenting path algorithm avoids its worst case behavior by doing <u>randomized greedy pre-processing</u> (randomized to avoid adversary test case) ☺, try this in VisuAlgo: <u>"Augmenting Path with Randomized Greedy Preprocessing", O(**kE**)</u>

# The ++ Code (1) ☺

```
vi match, vis; // global variables

int Aug(int L) { // return 1 if ∃ an augmenting path from L
  if (vis[L]) return 0; // return 0 otherwise
  vis[L] = 1;
  for (auto &R : AL[L])
    if (match[R] == -1 || Aug(match[R])) {
      match[R] = L;
      return 1; // found 1 matching
    }
  return 0; // no matching
}
```

# The ++ Code (2) ☺

```
int main() {
  ios::sync_with_stdio(false); cin.tie(NULL);

// inside int main()
  // build unweighted bipartite graph
  // with directed edge left->right set

  unordered_set<int> freeV;
  for (int L = 0; L < Vleft; L++) // assume all vertices
    freeV.insert(L);   // on left set are free initially
  match.assign(V, -1); // V is the number of vertices
  int MCBM = 0;        // in bipartite graph
```

# The ++ Code (3) ☺

```
// Greedy pre-processing for trivial Augmenting Paths
for (int L = 0; L < Vleft; L++) { // O(V^2)
   vi candidates;
   for (auto &R : AL[L])
     if (match[R] == -1)
       candidates.push_back(R);
   if (candidates.size() > 0) {
     MCBM++;
     freeV.erase(L); // L is matched, no longer a free vertex
     int a = rand()%candidates.size(); // randomize this
     match[candidates[a]] = L;         // greedy matching
   }
 }
```

# The ++ Code (4) ☺

```
  // for each of the k remaining free vertices
  for (auto &f : freeV) {
    vis.assign(Vleft, 0); // reset before each recursion
    MCBM += Aug(f); // once f is matched,
  }                 // f remains matched till end
  cout << "Found " << MCBM << " matchings\n";

  return 0;
}
```

# Live Solve

https://open.kattis.com/problems/b

From my Augmenting Path

Algorithm++ baseline code (C++)

Solution:

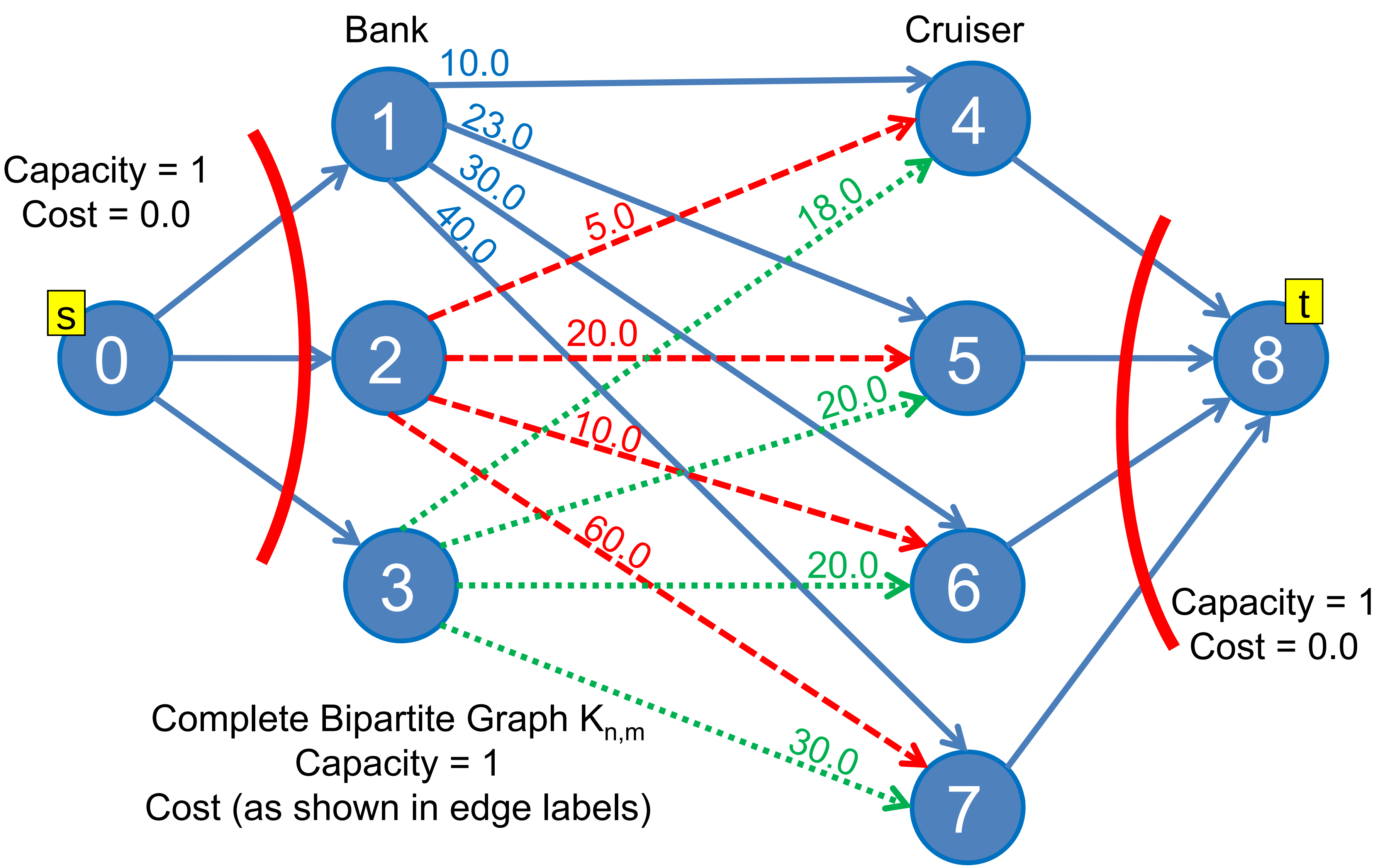
Min/Max Cost (Max) Flow (Overview Only)

Kuhn Munkres's (Hungarian, Overview Only)

# WEIGHTED MCBM

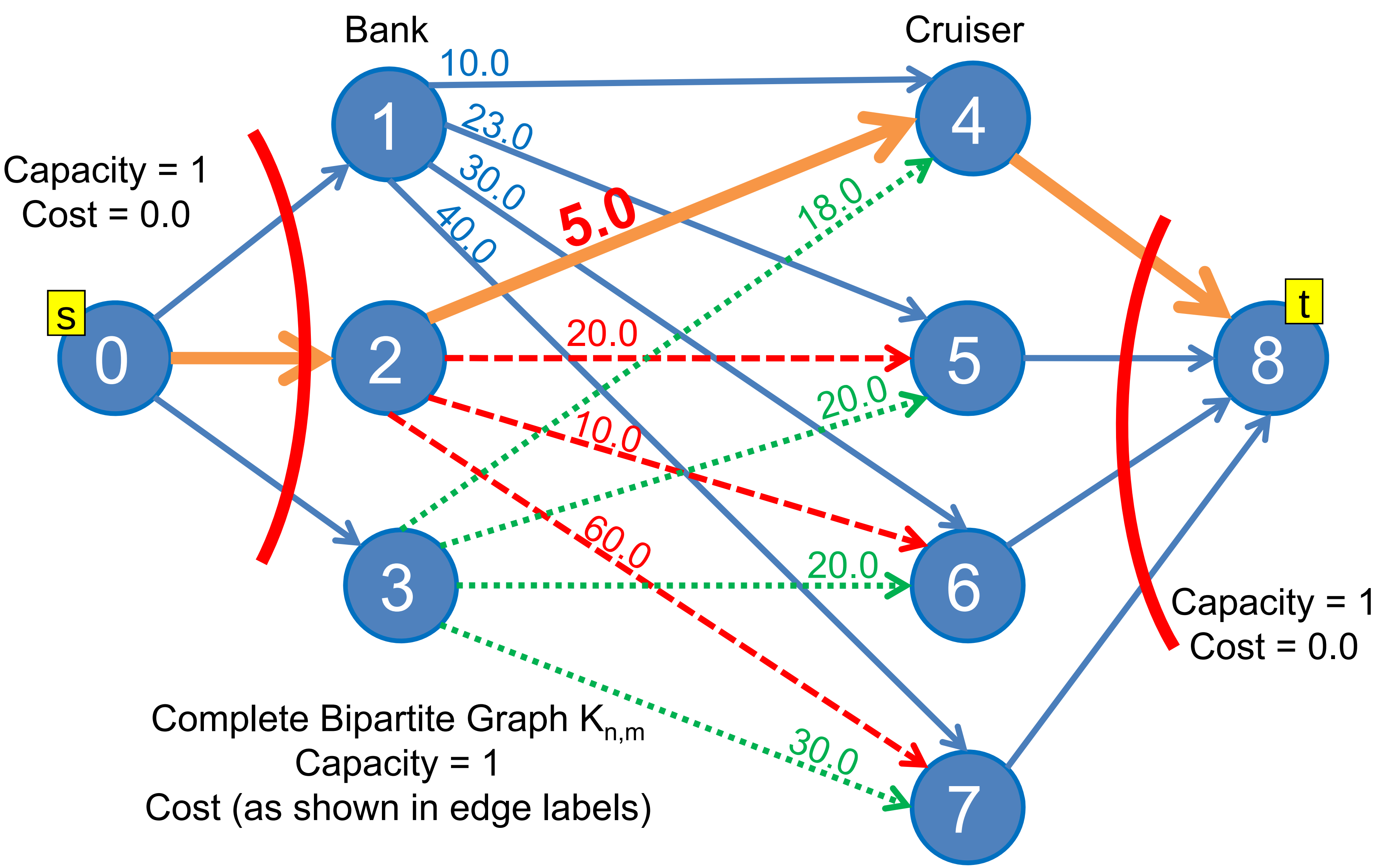
# UVa 10746 (1)

Find Max Flow (=3 here)
of Minimum Cost (still in **P**)
That is, Min Cost Bipartite Matching…



# UVa 10746 (2)

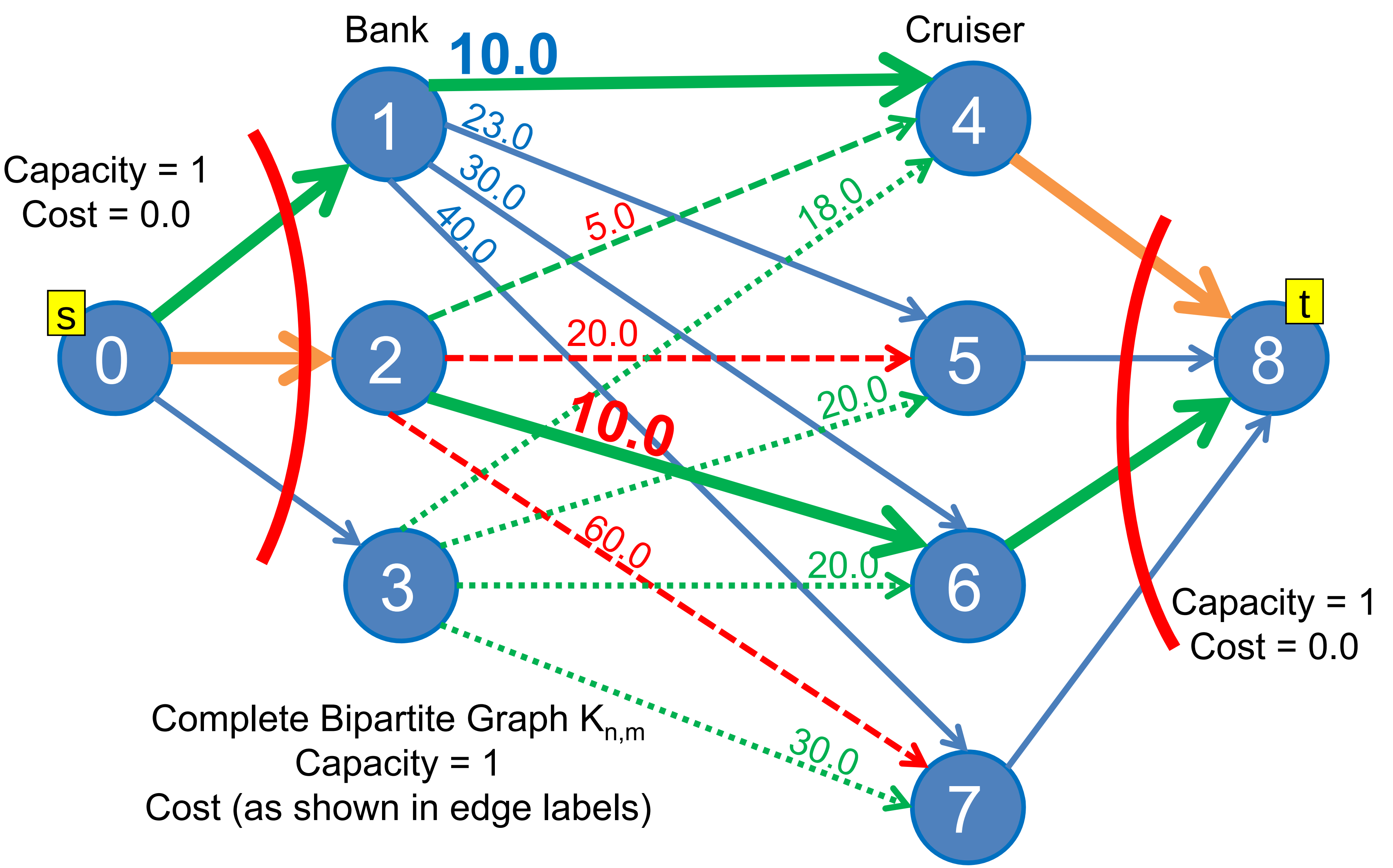
Min Cost so far =
0 + 5.0 + 0 = 5.0




Complete Bipartite Graph $K_{n,m}$
Capacity = 1
Cost (as shown in edge labels)

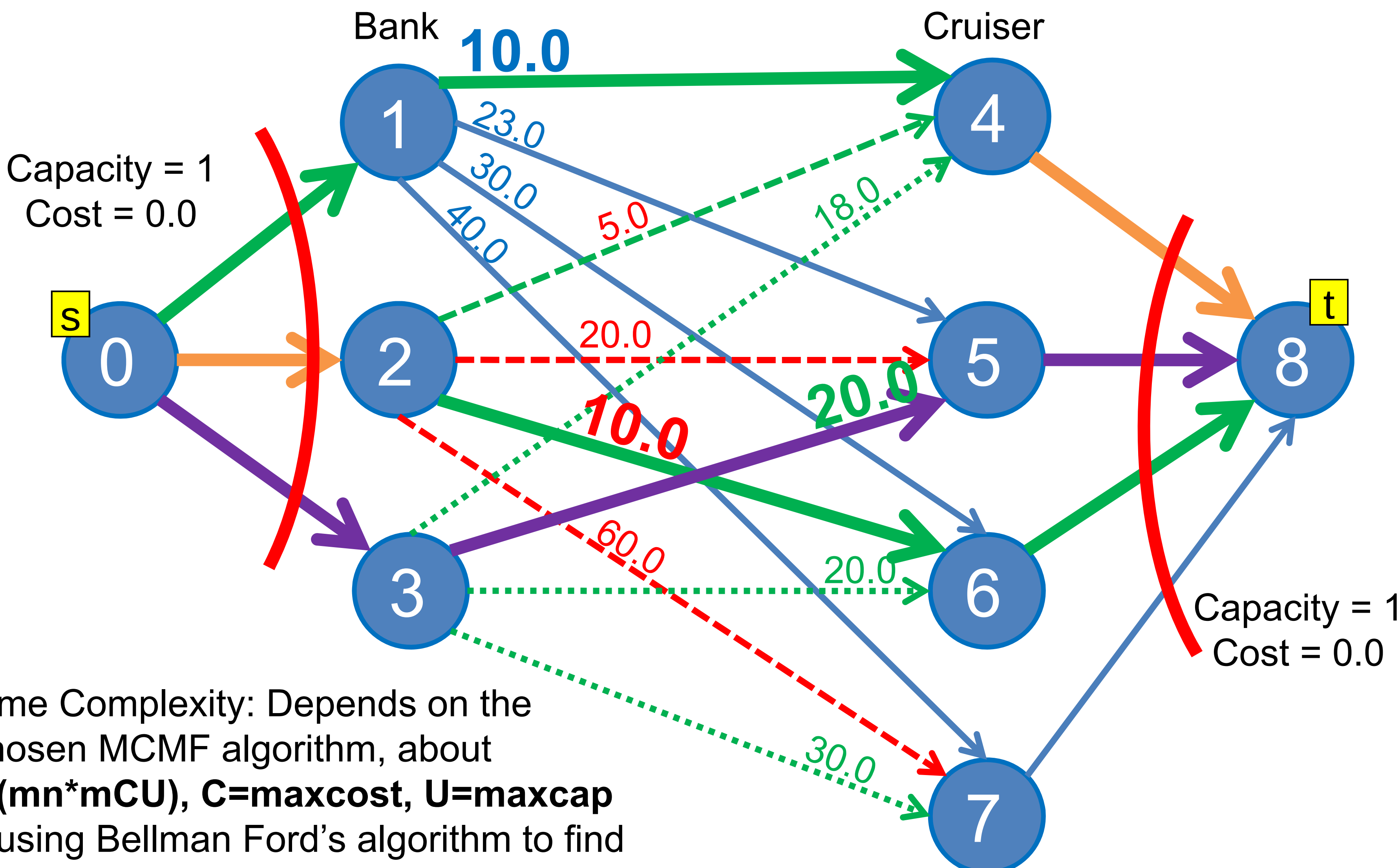
# UVa 10746 (3)

Min Cost so far =
0 + **5.0** + 0 +
0 + 10.0 **- 5.0** + 10.0 + 0 = 20.0



# UVa 10746 (4)

Min Cost so far =
0 + **5.0** + 0 +
0 + 10.0 **- 5.0** + 10.0 + 0 +
0 + 20.0 + 0 = 40.0




Time Complexity: Depends on the chosen MCMF algorithm, about **O(mn*mCU), C=maxcost, U=maxcap** if using Bellman Ford's algorithm to find the cheapest augmenting path… (bad…)

# Kuhn Munkres's (Hungarian) Algorithm

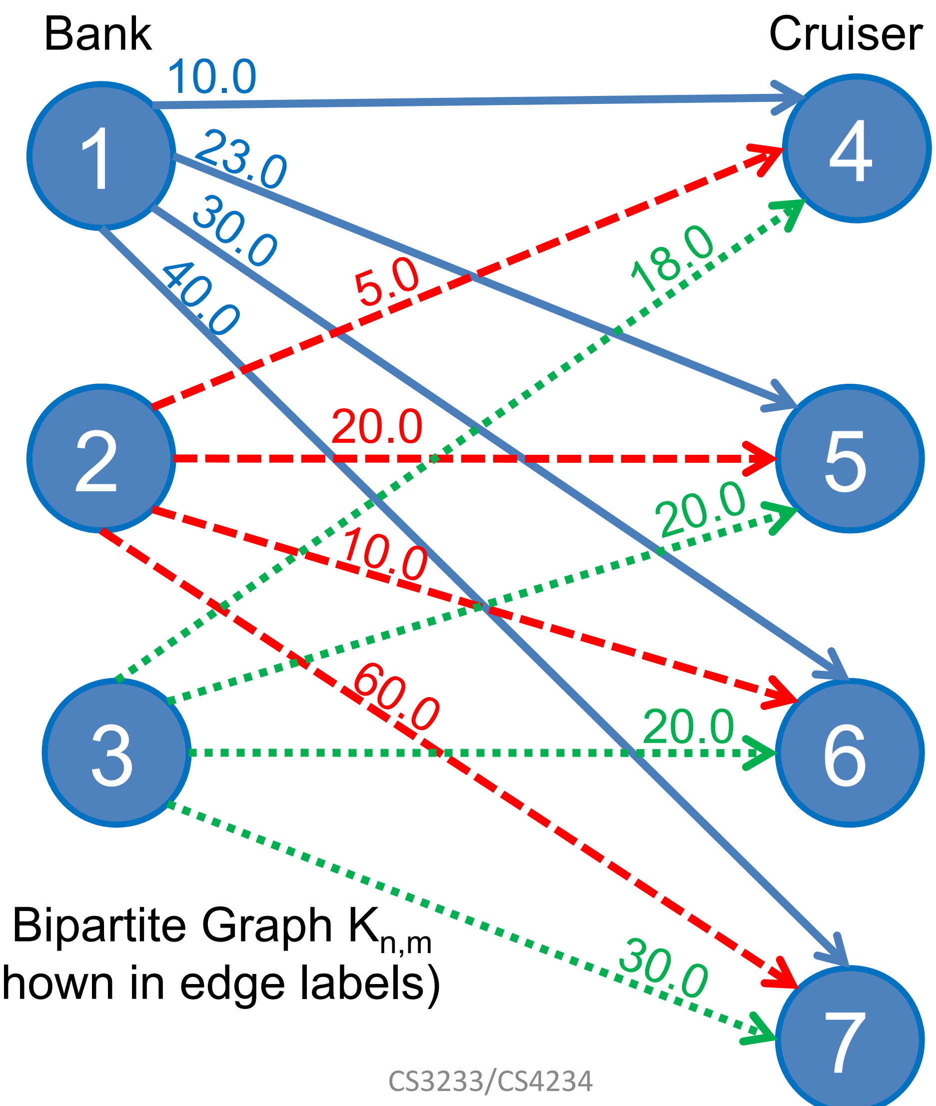
Harold Kuhn and James Munkres name their algorithm based on the work of two other Hungarian mathematicians (Denes Konig and Jano Egervary)

A rough explanation using this *other* animation tool:

https://www-m9.ma.tum.de/graph-algorithms/matchings-hungarian-method/index_en.html

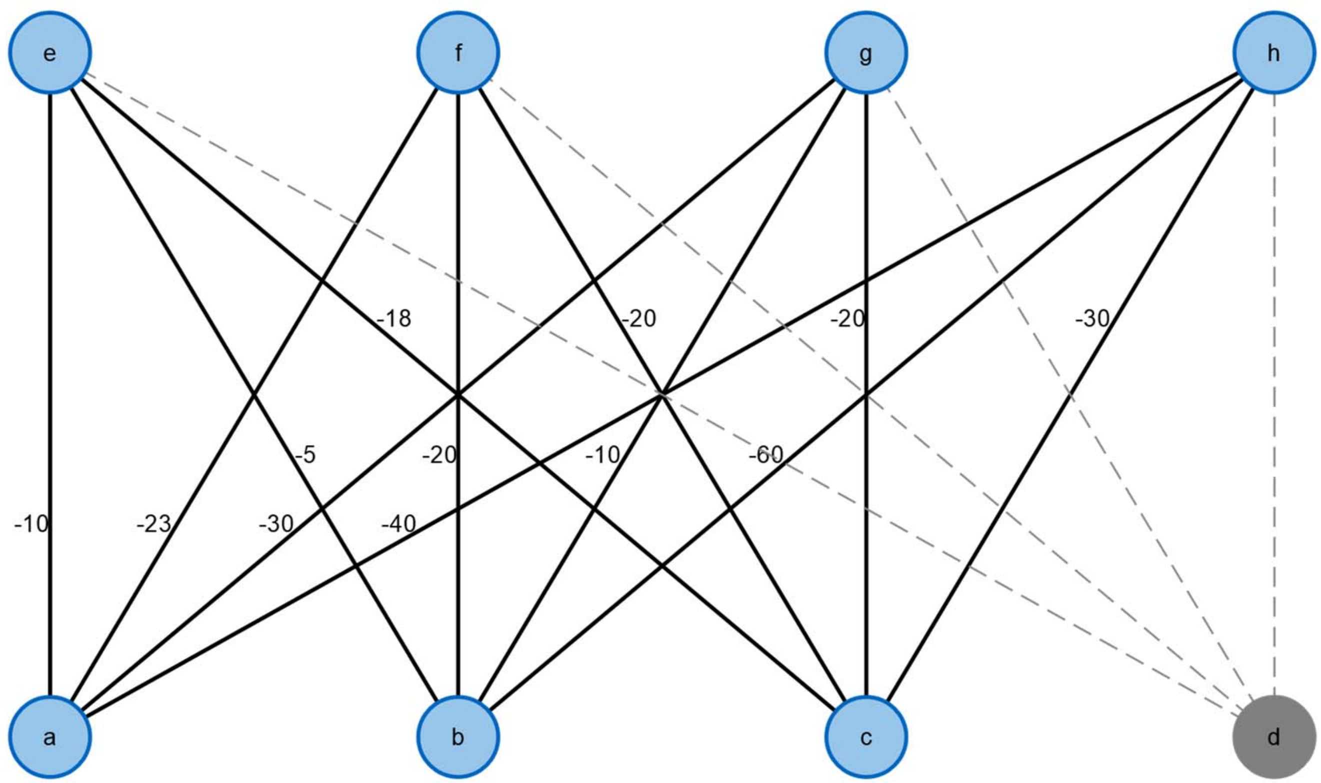
# UVa 10746 (5)

Find MCBM (=3 here)
of <u>Minimum Cost</u> (still in **P**)
That is, Min Cost Bipartite Matching…




Complete Bipartite Graph $K_{n,m}$
Cost (as shown in edge labels)

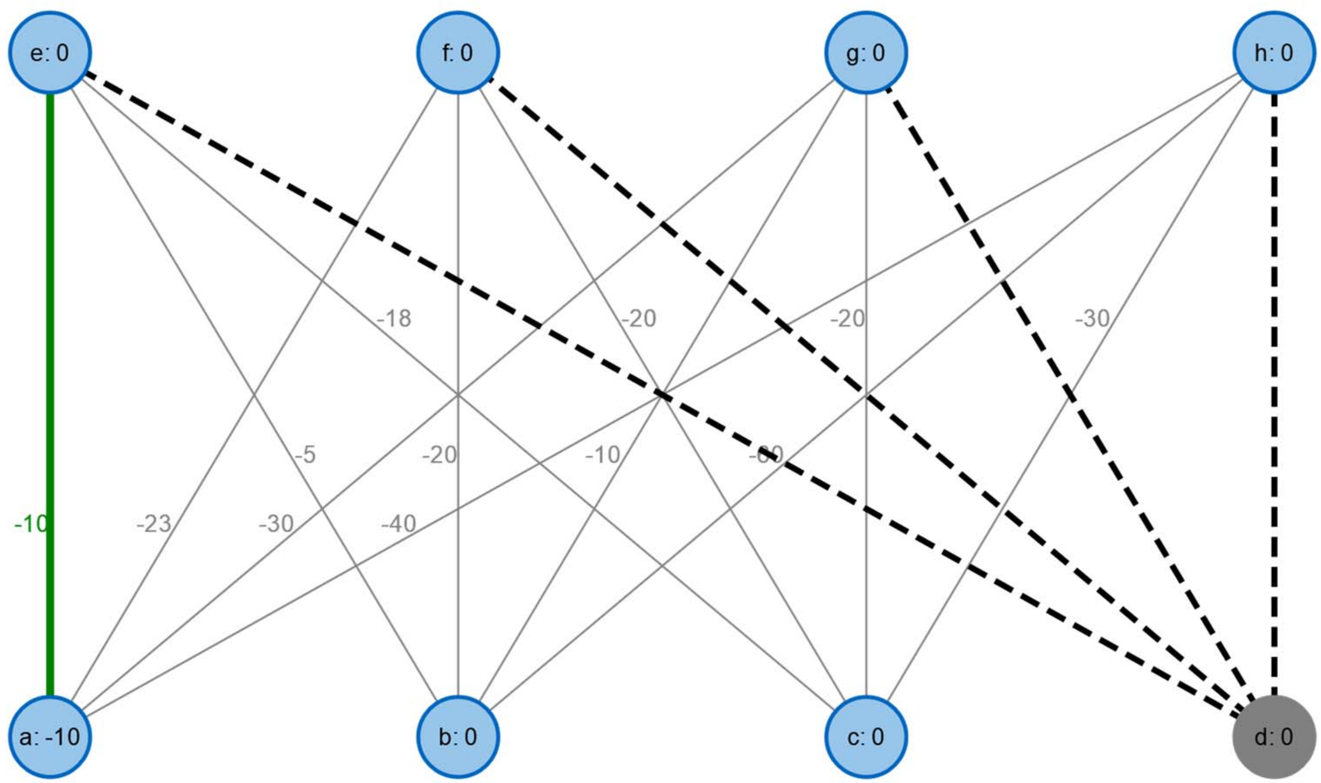
# UVa 10746 (6)




This tool https://www-m9.ma.tum.de/graph-algorithms/matchings-hungarian-method/index_en.html is on maximization problem on complete bipartite graph, so I have to convert the weights to negative and add dummy vertex/edges to make $K_{4,4}$

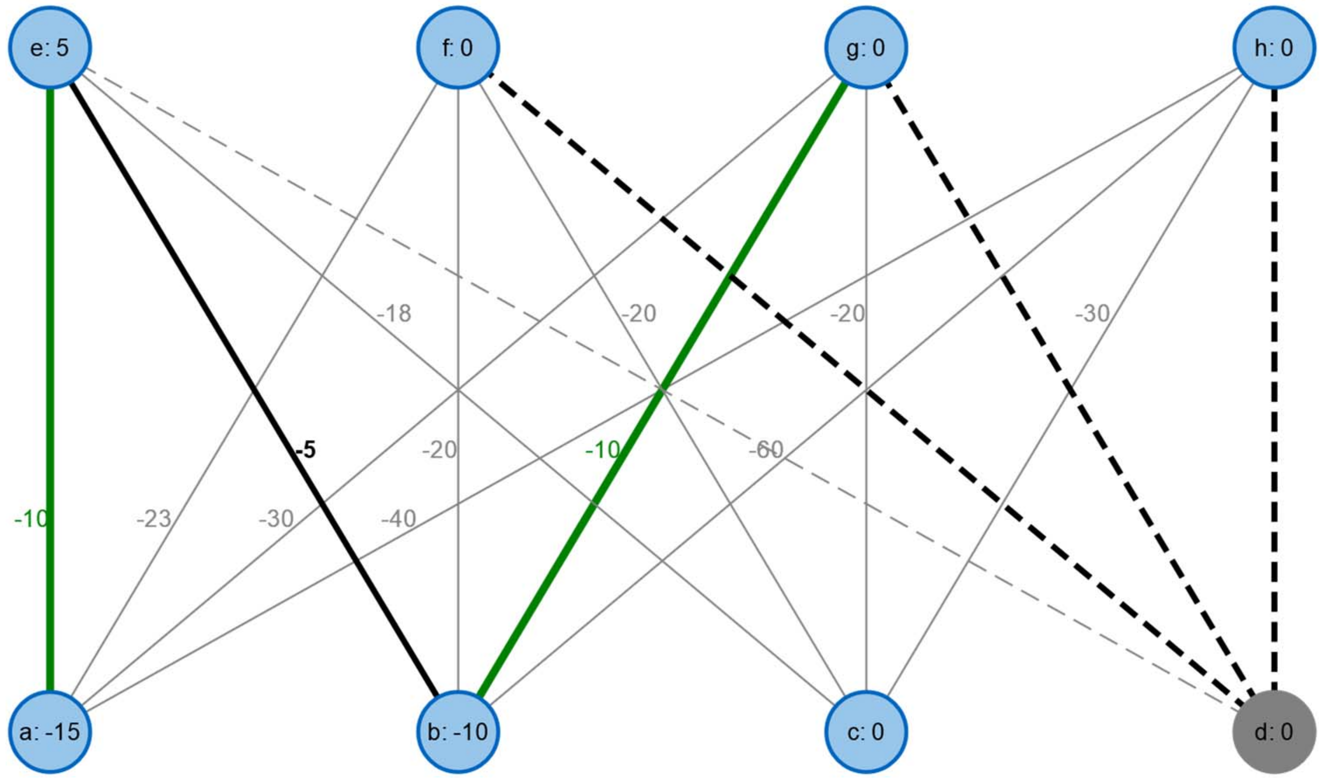
The layout is top row = right set and bottom row = left set, vertex ‘d’ is ‘virtual’

# UVa 10746 (7)



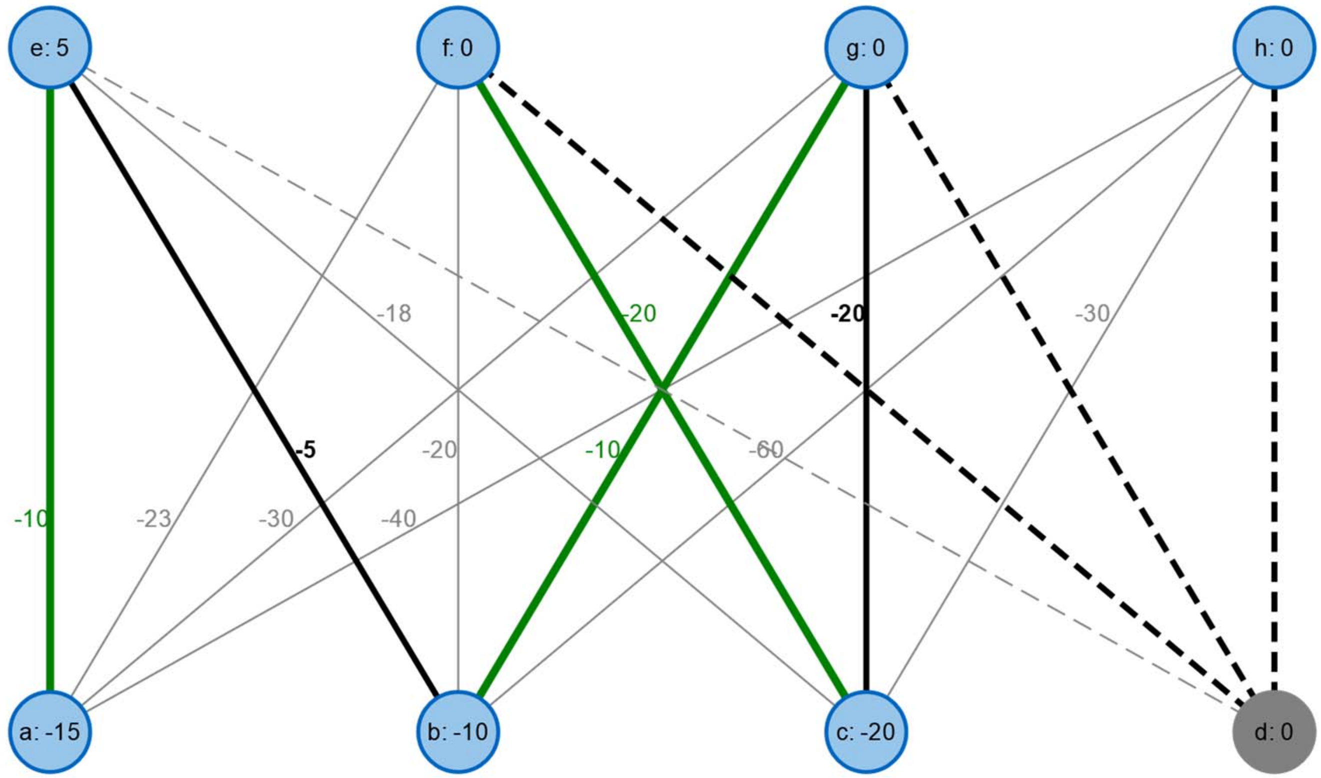

First matching with cost 10 (-10)

# UVa 10746 (8)



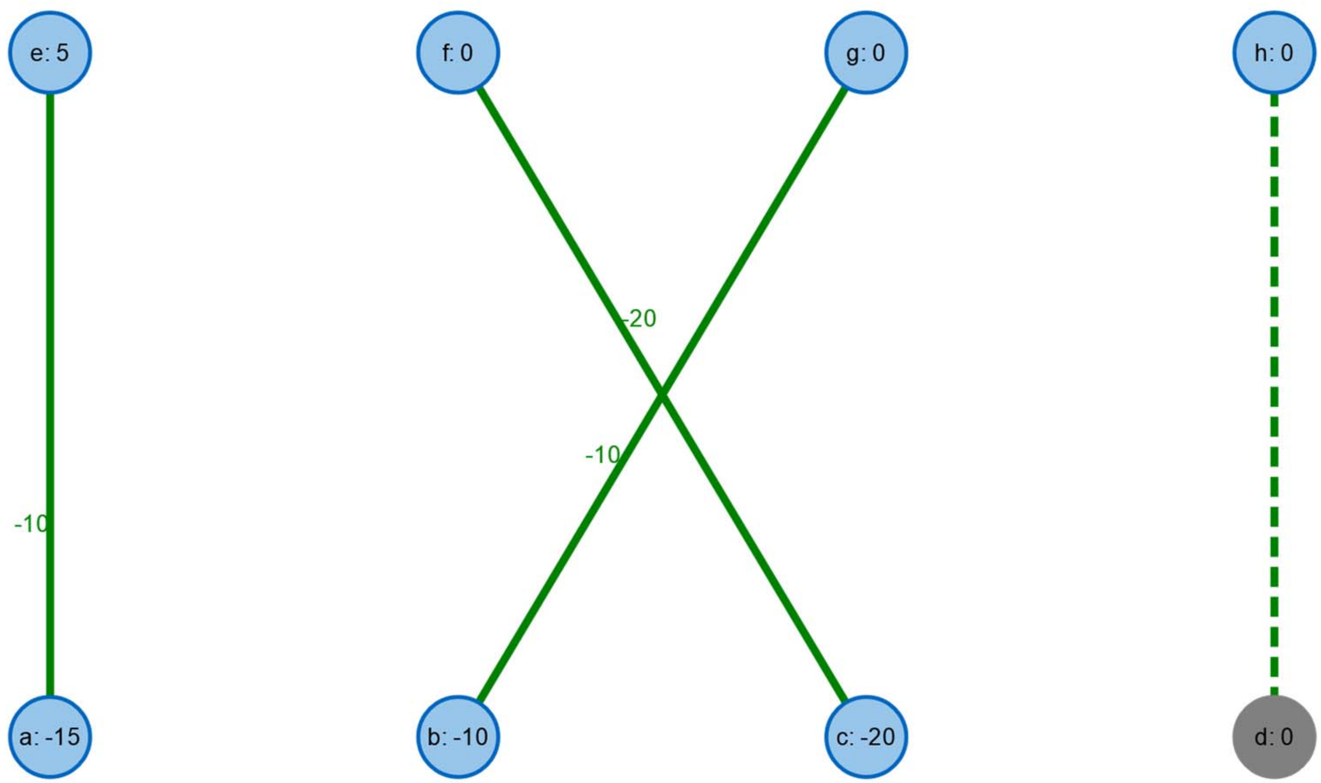

Second matching with cost 10 (-10) too (total 10+10 = 20) (-20)

# UVa 10746 (9)




Second matching with cost 20 (-20) too (total 10+10+20 = 40) (-40)

# UVa 10746 (10)




The last matching is just a dummy

# Kuhn Munkres's (Hungarian) Algorithm

A good Hungarian algorithm implementation runs in $O(\mathbf{V}^3)$, thus it is a much better algorithm for **Weighted MCBM** problem

- You are allowed to quote this info verbatim in exam
  - Focus on the modeling of the *complete* weighted bipartite graph
  - And on whether it is a maximization or a minimization problem
- References:
  - https://github.com/jaehyunp/stanfordacm/blob/master/code/MinCostMatching.cc
  - http://e-maxx.ru/algo/assignment_hungary
  - https://brilliant.org/wiki/hungarian-matching/

Solution:

Edmonds's Matching Algorithm (Overview only)

DP with Bitmask (Small Graph, discussed later)

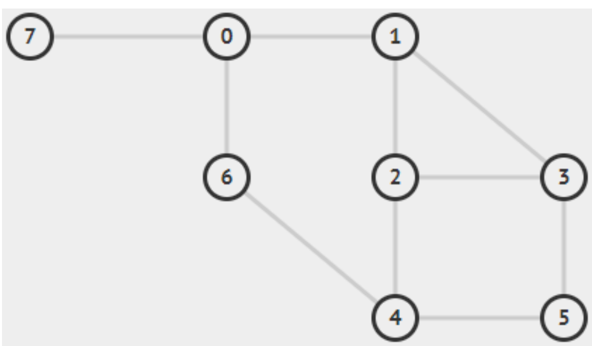
# UNWEIGHTED MCM

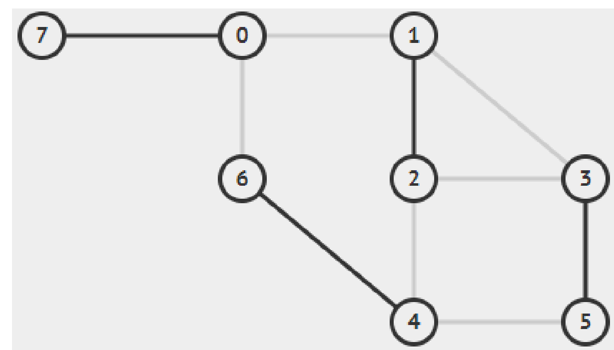
[https://visualgo.net/en/matching](https://visualgo.net/en/matching), select “Unweighted General” tab

# Non-Bipartite Graph and Blossom

A graph is not bipartite if it has at least one odd-length cycle

What is the MCM of this non-bipartite graph?




It is harder to find augmenting path (Berge’s Lemma) in such graph due to alternating cycles called **blossoms**

# Blossom Shrinking/Expansion

Shrinking these blossoms (recursively)
will make this problem "easy" again

# When To Use Edmonds's Matching?

This algorithm is a bit hard to implement…

O($\mathbf{V}^3$) <u>library code</u> is preferred

- Only used for unweighted MCM with $\mathbf{V} \in$ [22..200*; complex code]
  - If $\mathbf{V} \leq$ [19..21], see [this]
- Randomized greedy pre-processing is ALSO APPLICABLE here!!
  - Again, you can quote this verbatim for final assessment
  - Focus on the unweighted non-bipartite graph modeling

For code, just use external source

- http://codeforces.com/blog/entry/49402 (C++)
- https://sites.google.com/site/indy256/algo/edmonds_matching (Java)

Solution(s):

DP with Bitmask (only for small graph)

Edmonds' Matching (future work…)

# WEIGHTED MCM

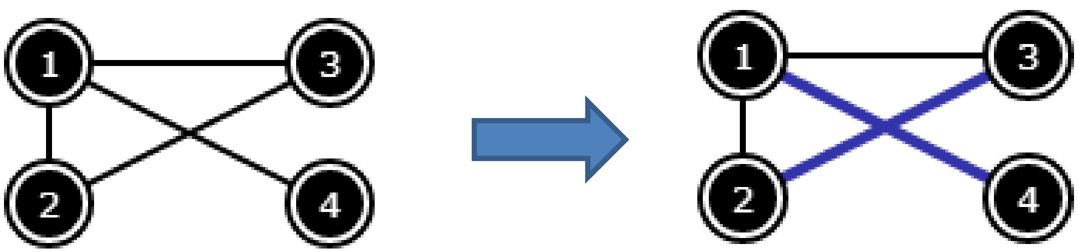
# No? Choice… (only for **V** ≤ [19..21])

```
ii wMCM(int mask) {
  if (mask == (1<<N)-1) return ii(0, 0);
  if (memo[mask] != ii(-1, -1)) return memo[mask];
  int p1, p2;
  for (p1 = 0; p1 < N; p1++) if (!(mask & (1<<p1))) break;
  ii ans = wMCM(mask | (1<<p1)); // p1 unmatched
  for (p2 = p1+1; p2 < N; p2++)
    if (!(mask & (1<<p2)) && cost[p1][p2]) {
      ii nxt = wMCM(mask | (1<<p1) | (1<<p2)); // match p1-p2
      nxt.first += 2; nxt.second += cost[p1][p2];
      if ((nxt.first > ans.first) || // equal # matching
          ((nxt.first == ans.first) &&
           (nxt.second < ans.second))) // or smaller cost
        ans = nxt;
    }
  return memo[mask] = ans;
}
```

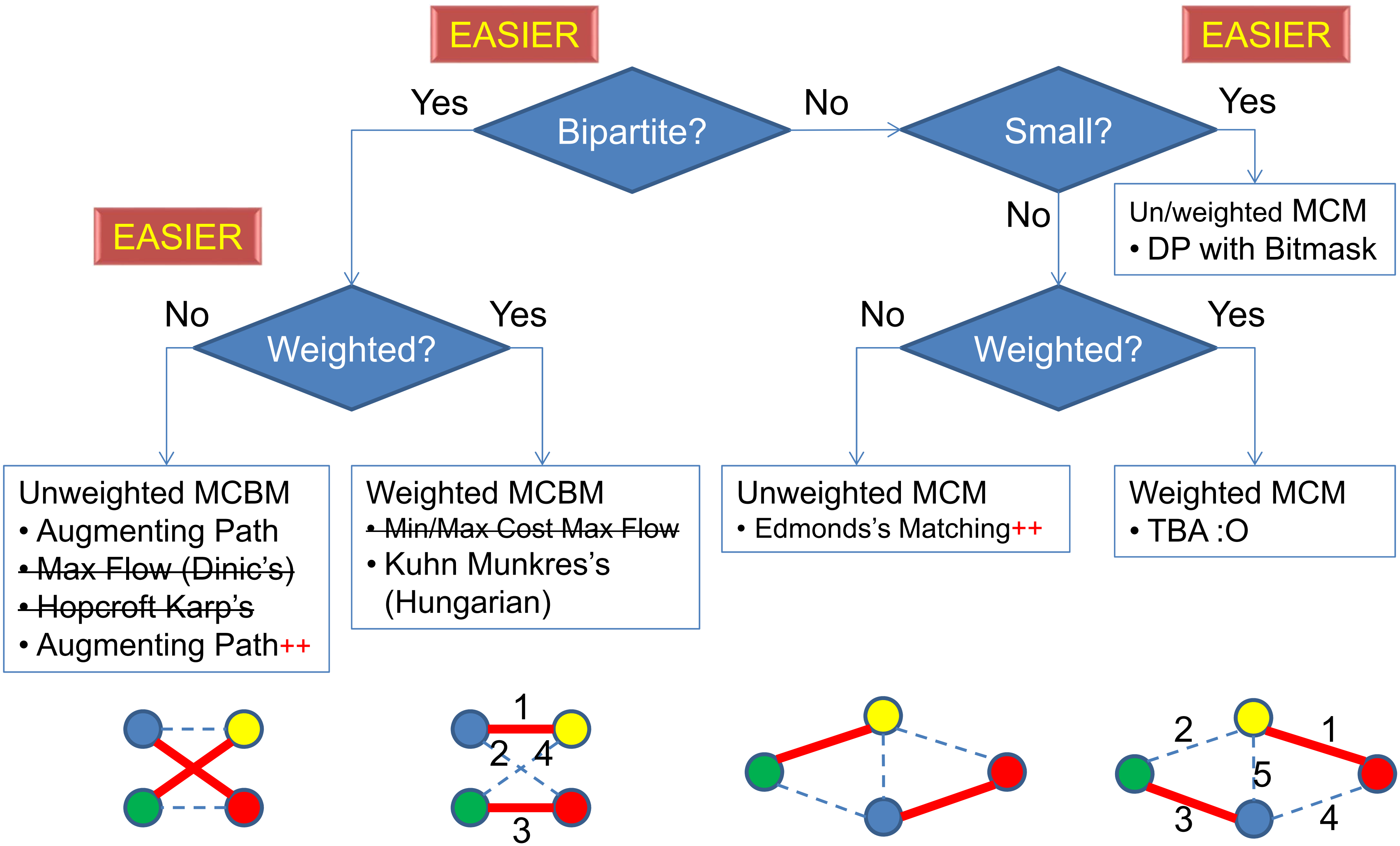
https://visualgo.net/en/recursion, select the “Matching” example…

# Also for Small Unweighted MCM Too

Just set all weights = 1 in the previous code



# Summary (so far…)



# References

- Mostly CP3.17b, Section 4.7, 4.8.4, 9.9 (still empty, being updated), 9.13, 9.17 (still empty, being updated), and 9.21 ☺
- TopCoder PrimePairs, RookAttack solution
- http://www.comp.nus.edu.sg/~cs6234/2009/Lectures/Match-sl-PC.pdf (Prof LeongHW's/P Karras slides)

- More matching exercises/applications during T07