CSE 300

Technical Writing and Presentation Sessional

Report: Maximum Bipartite Matching

Lab Section - A1

Group - 09

9th March, 2024

Members of the Group:

i. 2005012 - Abrar Jahin Sarker

ii. 2005017 - Abdullah Muhammed Amimul Ehsan

iii. 2005020 - Mostafa Rifat Tazwar

Contents

1 Introduction

The concept of maximum bipartite matching, a fundamental problem in graph theory, holds significant relevance across diverse fields due to its ability to model various real-world scenarios. Bipartite graphs serve as the foundational structure for this problem. In essence, a maximum bipartite matching seeks to pair elements from one subset with elements from the other subset in such a way that maximizes the number of paired elements. From applications in network flow optimization, job scheduling, and medical residency programs to its role in solving matching problems in societal contexts like marriage and kidney exchange programs, the utility of maximum bipartite matching extends across numerous domains, making it a subject of continued research and innovation.

1.1 Definitions

1.1.1 Bipartite Graph

A bipartite graph is one whose vertices can be split into two independent groups U,V such that **every edge connects vertices of different groups.**

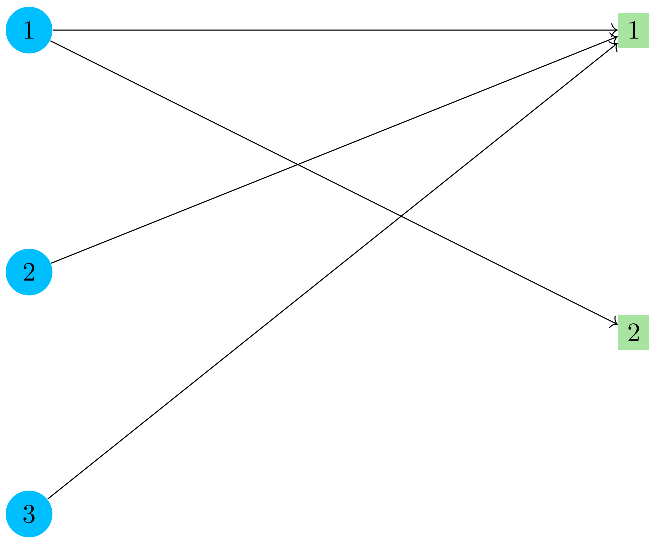
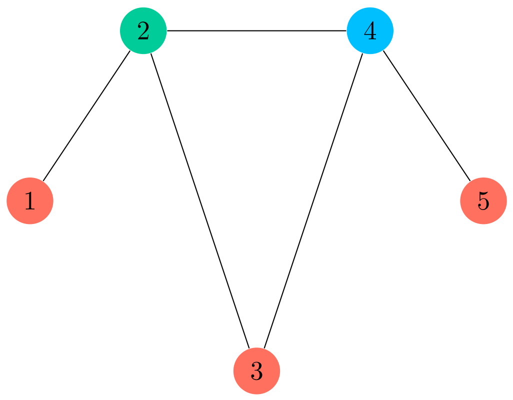


Figure 1: Visualization of bipartite graph

Properties of a Bipartite graph

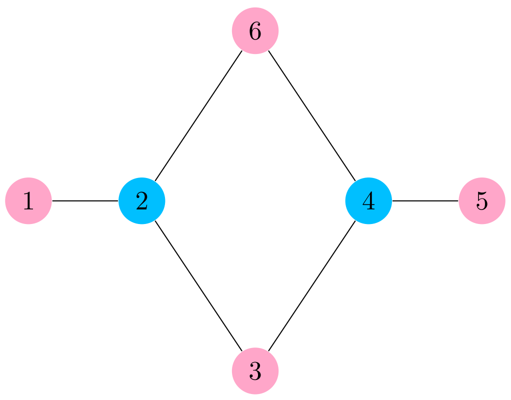
- There can not be any edge between any two vertices of U or any two vertices of V .
- The graph is two colourable and doesn't have cycles of odd length.



Odd length cycle

Not 2 colorable

Not a bipartite graph



Even length cycle

2 colorable

Bipartite graph

Figure 2: Comparison between Bipartite and Non-bipartite graph

1.1.2 Maximum Matching

Given a bipartite graph, a matching is a subset of the edges for which **every vertex belongs to exactly one of the edges**.
And, a maximum matching is a matching of **maximum number of edges**. In a maximum matching, if any edge is added to it, it is no longer a matching.

2 The Problem

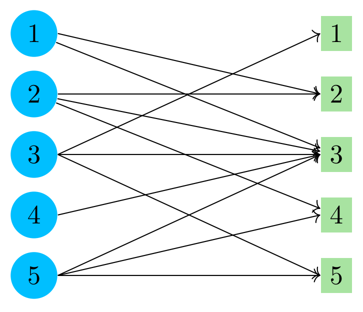
Given a bipartite graph $G = (A \cup B, E)$*, find an* $\{ S \subseteq A \times B : S$ *is a matching and is as large as possible.* $\}$ [1]

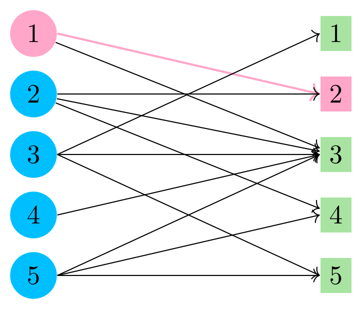
3 Scenario

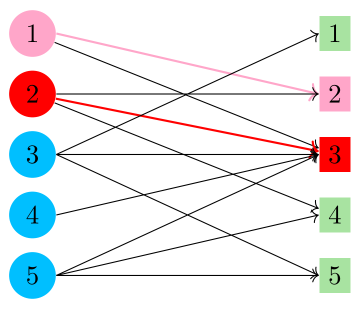
In a picnic,there are 5 people and 5 food items.Some people express interest in some of the items.How can we satisfy **maximum number of people** while wasting **minimum number of items**?

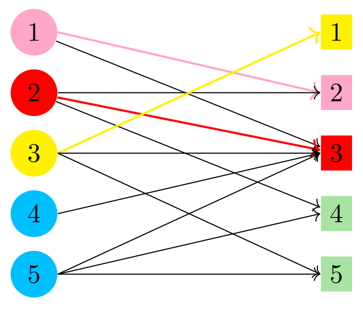
4 Greedy Approach

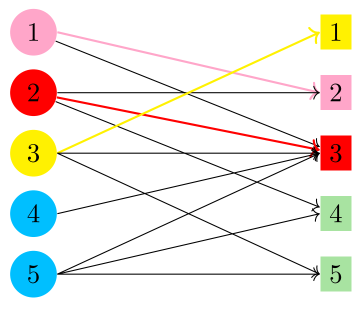
Greedy approach seeks to find the **first available item** among the items in which the person is interested in and assigns that item to the person. Here's a simulation for the above scenario :-

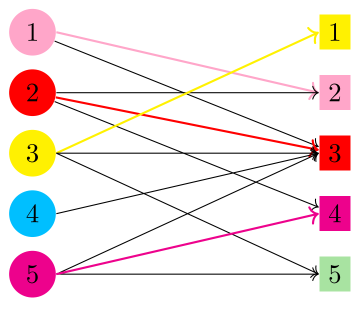

(a)


(b)


(c)


(d)


(e)


(f)

a) We define the problem using a graph, where any blue vertex represents a person to whom no item has been assigned and any green vertex represents an available item. Edges from a person to an item depicts their interest in that item.
b) Person 1 initiates the process by selecting the first available item (item 2).
c) Person 2's only viable option is item 3, as item 2 has already been chosen.
d) Person 3 selects item 1.
e) Person 4 cannot be assigned an item, as all options previously chosen by them have been allocated.
f) Person 5 discovers that item 4 is available for selection.

Bipartite matching : {(1, 2), (2, 3), (3, 1), (5, 4)}
Although greedy provides us with a valid matching , it is not maximal.

5 Reduction to a Known Problem

- Given an instance of bipartite matching
- Reduce it to Max Flow Problem.
- Where the solution to the network flow problem can easily be used to find the solution to the bipartite matching.

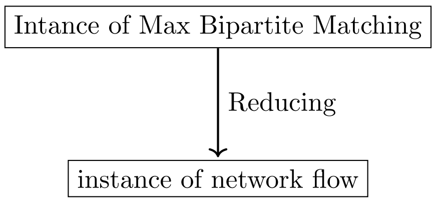


Figure 3: Approach to solve

6 Steps of Reduction

6.1 Make all the edges directed

Make all the edges directed if not and add 0 flow and 1 capacity for all edges, expressed as 0/1 in Flow/Capacity format.

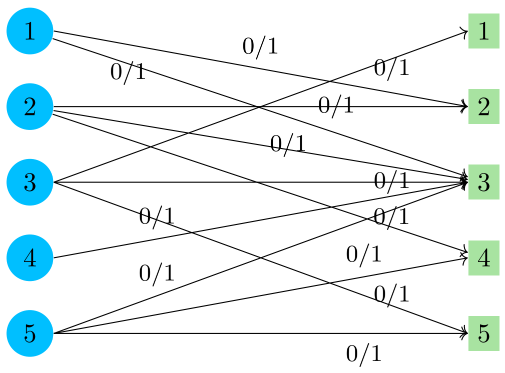


Figure 4: All edges are directed

6.2 Add two new nodes

Add two new nodes: Source (S) and Destination (t).These nodes are needed to obtain the common source node and the common destination/sink node which is needed for suitability to apply network flow algorithms.

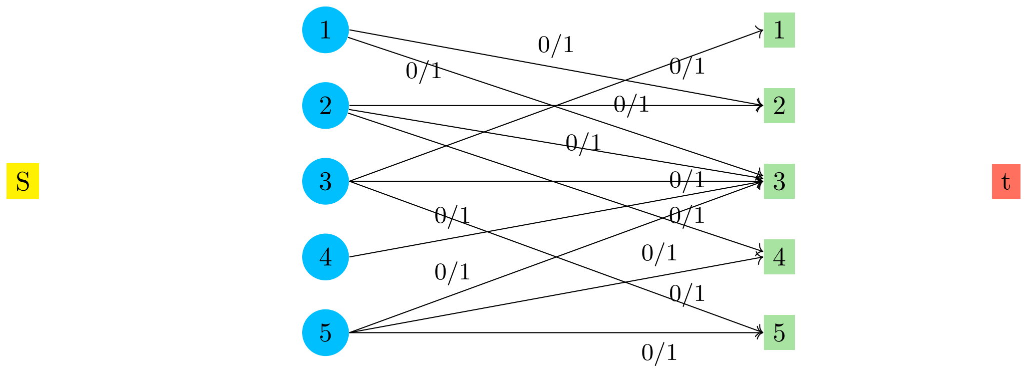


Figure 5: Two nodes (s and t) added

6.3 Add edges from source to person

Add edges from the source to each person with flow=0 and capacity=1.Here capacity =1 means every person is allowed to take only one more item.Flow=0 means currently nobody has taken any item.

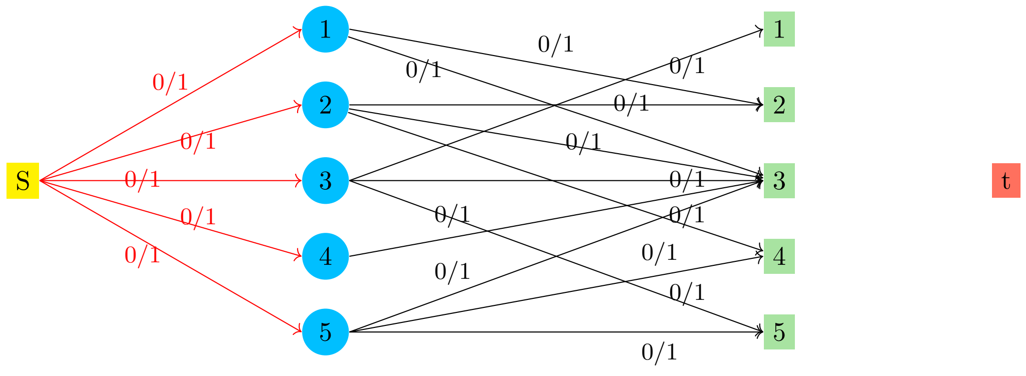


Figure 6: edges added from source to person

6.4 Add edges from food items to destination

Add edges from food items to the destination with 0 flow and 1 capacity.Capacity is the number of copies remaining for a certain item and the flow denotes how many copies have already been taken.

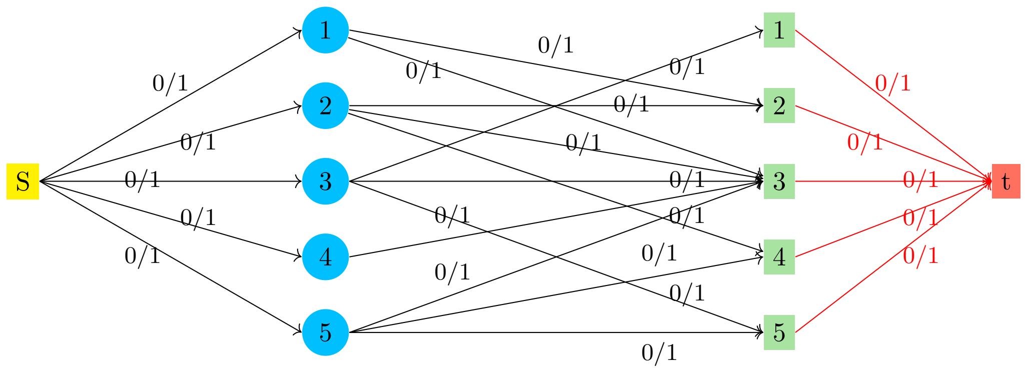


Figure 7: edges added from food items to sink

6.5 Reduced to Network Flow Problem

Finally, the problem is reduced to a network flow problem where we have to apply a network flow algorithm. Here, flow > 0 between the pairs indicates a matching.

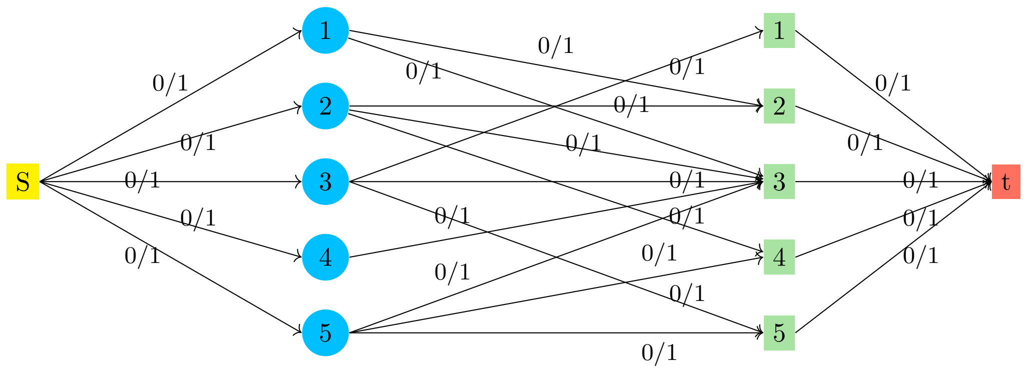


Figure 8: Ready to apply Max Flow Algorithm

7 Popular Algorithms for Maximum Flow Problem

1. **Ford-Fulkerson Algorithm:**
   - Uses the concept of augmenting paths to find the maximum flow.
   - Time complexity: $O(E \cdot \text{max\_flow})$, where E is the number of edges and max_flow is the maximum flow.
   - Space complexity: $O(V)$, where V is the number of vertices.
2. **Edmonds-Karp Algorithm (a variation of Ford-Fulkerson):**
   - Utilizes BFS (Breadth-First Search) to find augmenting paths.
   - Time complexity: $O(VE^2)$, where V is the number of vertices and E is the number of edges.
   - Space complexity: $O(V^2)$, where V is the number of vertices.
3. **Dinic's Algorithm:**
   - Builds a layered network and performs multiple blocking flows.
   - Time complexity: $O(V^2 \cdot E)$, where V is the number of vertices and E is the number of edges.
   - Space complexity: $O(V^2)$, where V is the number of vertices.
4. **Push-Relabel Algorithm:**
   - Maintains a preflow that satisfies certain conditions and iteratively pushes flow along edges.
   - Time complexity: $O(V^3)$, where V is the number of vertices.
   - Space complexity: $O(V^2)$, where V is the number of vertices.
5. **Goldberg-Tarjan Algorithm:**
   - A combination of push-relabel and relabel-to-front techniques.
   - Time complexity: $O(V^2 \cdot E)$, where V is the number of vertices and E is the number of edges.
   - Space complexity: $O(V^2)$, where V is the number of vertices.

Among all of them,Edmonds-Karp is easier to understand and very widely used .Here,We are using Edmonds-Karp algorithm to simulate max flow problem's solution

8 Edmonds-Karp Algorithm to Solve Network Flow Problem

The Edmonds-Karp algorithm is a popular method for finding the maximum flow in a network. It is based on the Ford-Fulkerson method and uses breadth-first search (BFS) to find augmenting paths efficiently.

Algorithm 1 Edmonds-Karp Algorithm

```
Require: Network graph $G$, source node $s$, sink node $t$
Ensure: Maximum flow $f$
 1: Initialize flow $f(u, v) = 0$ for all edges $(u, v)$ in the graph.
 2: while augmenting paths exist do
 3:     Use BFS to find the shortest augmenting path from $s$ to $t$.
 4:     if no augmenting path is found then
 5:         Terminate the algorithm.
 6:     end if
 7:     Let $P$ be the augmenting path found by BFS.
 8:     Let $cf(P)$ be the minimum residual capacity along path $P$.
 9:     for each edge $(u, v)$ in $P$ do
10:         Update flow $f(u, v) = f(u, v) + cf(P)$.
11:         Update flow $f(v, u) = f(v, u) - cf(P)$ for the reverse edge.
12:     end for
13: end while
14: The value of the maximum flow is the sum of flow values leaving the source.
```

This algorithm efficiently finds the maximum flow in a network by iteratively augmenting flow along the shortest augmenting paths. It terminates when no more augmenting paths can be found, guaranteeing the maximum flow is achieved.

9 Solution

As the algorithm uses BFS,the augmenting paths are not necessarily unique.Storing edges in different configurations may result in different valid solutions.Here we are trying to look for one of the valid solutions:

Augmenting path 1

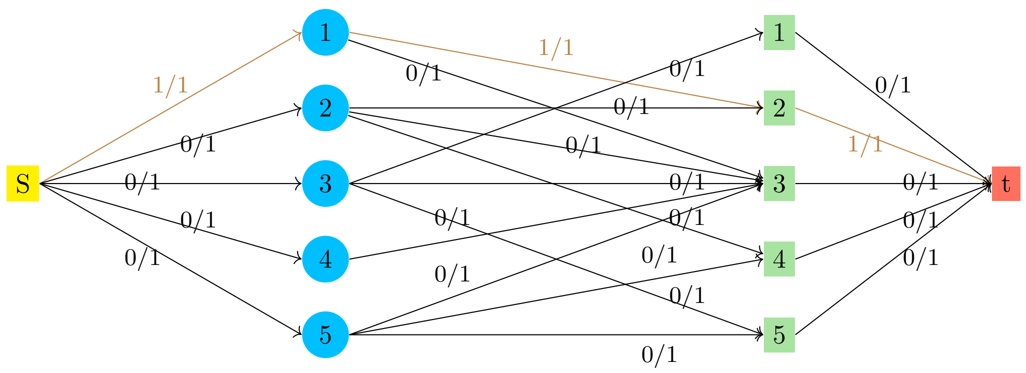


Figure 9: Found augmenting path -1

Here,the flow has become 1 along the path.The bottleneck capacity=1 along the path.As a result,remaining capacity has become $1 - 1 = 0$ along the path.Therefore,the edges along the path will not be selected by BFS applied later unless any backward flow is generated.

Augmenting path 2

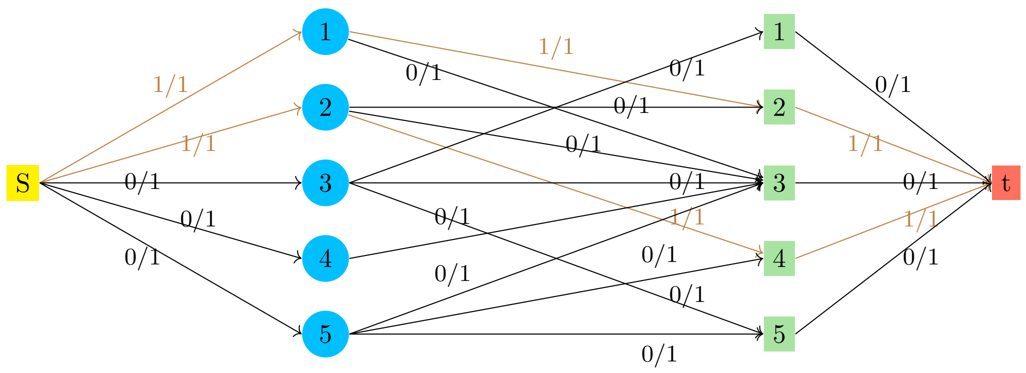


Figure 10: Found augmenting path -2

Similarly,we have found another augmenting path.Here,the bottleneck capacity is also 1.So the edges along the path are blocked for further usage.

Augmenting path 3

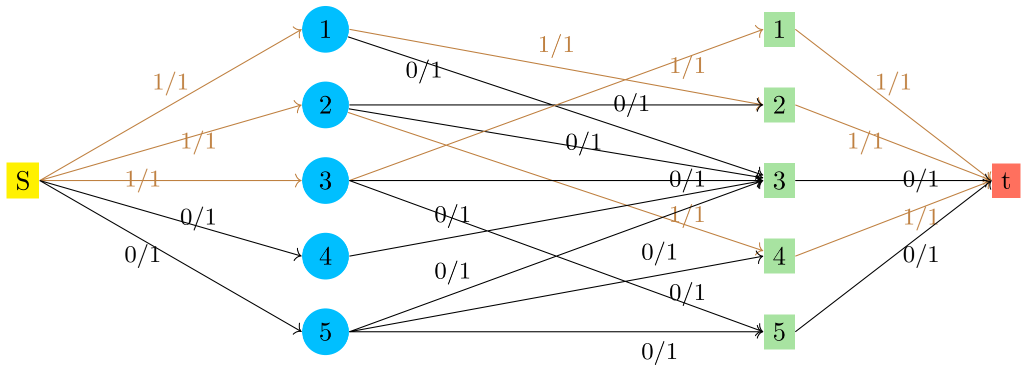


Figure 11: Found augmenting path -3

This is continuation of previous steps and this step is also handled accordingly

Augmenting path 4

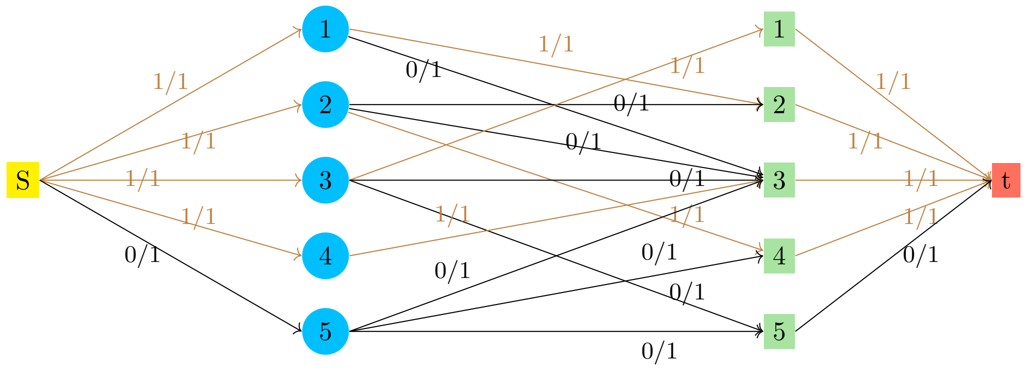


Figure 12: Found augmenting path -4

This process continues to find augmenting path as long as there is a path with remaining capacity greater than 0.

Augmenting path 5

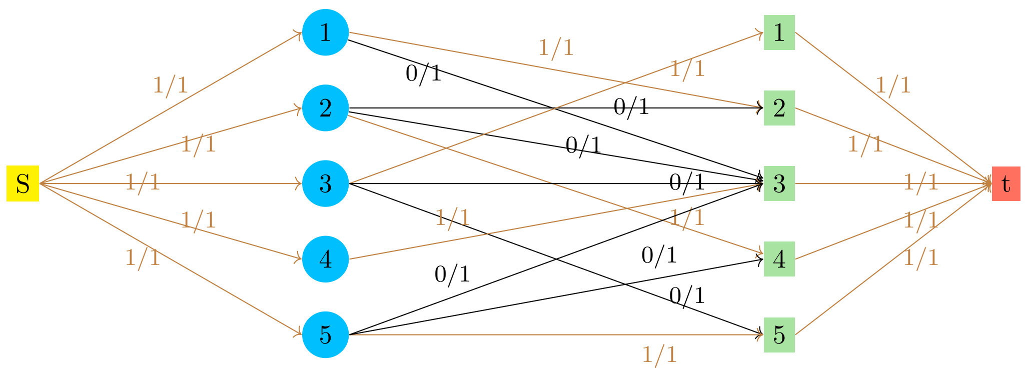


Figure 13: Found augmenting path -5,no more augmenting path possible

As we can see,we have finally found all possible augmenting paths.There is no more path available with remaining capacity >0 .
Here,the quantity of maximum flow can be obtained by two ways:

- Summation of flow leaving the source node s:

$$\sum_{v \in V} f(s, v)$$

  where V is the set of vertices and $f(u, v)$ denotes the flow from vertex u to vertex v.

- Summation of flow entering the sink (destination) node t:

$$\sum_{v \in V} f(v, t)$$

  where V is the set of vertices and $f(u, v)$ denotes the flow from vertex u to vertex v.

Here,we find it to be equal to 5. It implies we have become able to match 5 pairs.

10 Analysing the solution

Here,we can not augment the paths anymore.We have already found the max flow of the network.The max flow=5.Hence,Number of matching=5.
Pairs are :

1. Person 1,Item 2
2. Person 2,Item 4
3. Person 3,Item 1
4. Person 4,Item 3
5. Person 5,Item 5

Using greedy approach,we found 4 pairs which is not maximum.
Thus,max flow problem helped us find an optimal solution in polynomial time.

11 Time Complexity

Edmond Karp's algorithm uses BFS as we need the shortest augmenting path from source to sink. Running time of BFS is $O(V + E)$. Path search can take upto $O(VE)$ time. Some calculation and reduction leads us to a time complexity of $O((V + E)VE) = O((E)VE) = O(VE^2)$.

12 Extension

Now some special cases will be discussed where the algorithm needs to be extended to solve the problem.

12.1 Allowing multiple items for a person

In this case we will make the capacity of the edges from source to person, equal to the number of items they can take. Then we can use the same algorithm to find the maximum matching.
For example if Person 2 is allowed to take 3 items and person 5 is allowed to take 2 items:

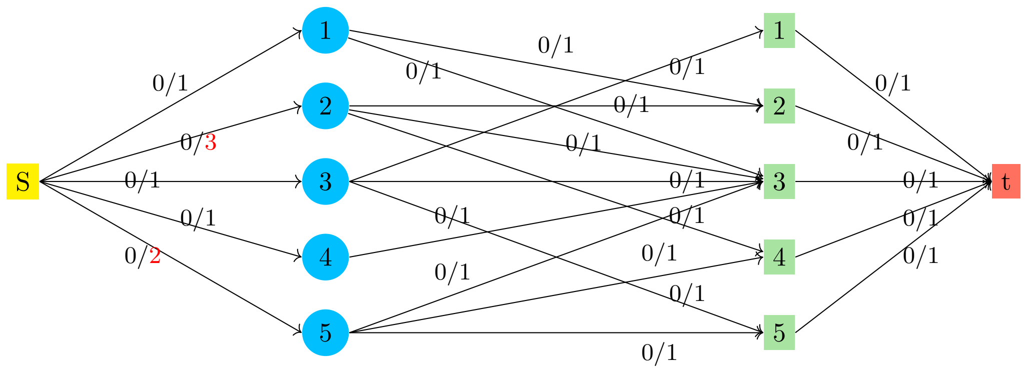


Figure 14: Allowing multiple item(s) for a person

12.2 Multiple copies of item available

This case almost similar to the previous one. We will make the capacity of the edges from item to sink, equal to the number of copies of the item available. Then we can use the regular network flow algorithm to find the maximum matching.
For example if item 1 has 3 copies and item 5 has 4 copies:

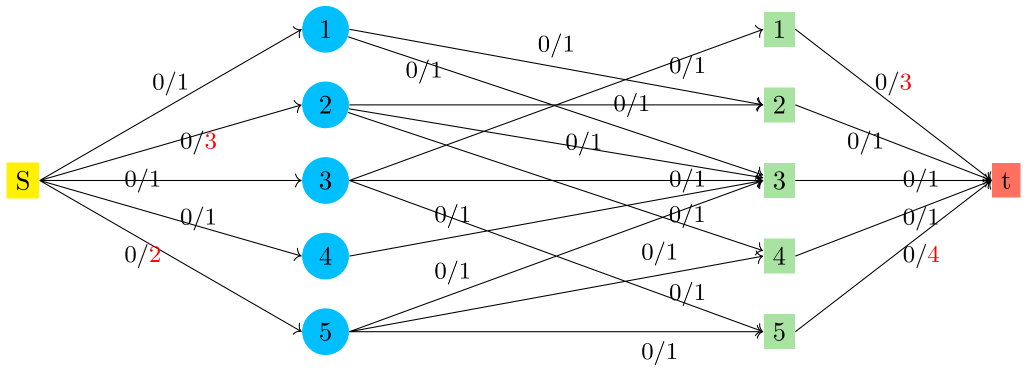


Figure 15: Multiple copies of items available

12.3 Taking same item multiple times

We have to make the edges from person to item have capacity, the number of times that particular person can take that particular item. Then we can use our network flow algorithm to solve the problem.
For example if Person1 can take Item 2 upto 5 times:

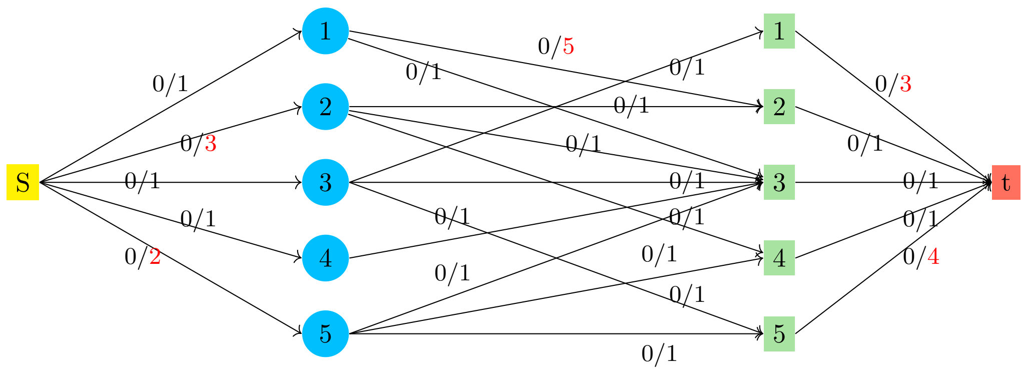


Figure 16: Allowed to take same item multiple times

13 References

References

[1] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.