Uma solução para o Linear Assignment Problem

Discente: João Victor de Alarcão Ayalla Alcântara

Um pouco sobre mim

- Sou o João Victor de Alarcão Ayalla Alcântara, porém sou mais conhecido como Ayalla.
- Ingressei na Universidade Federal de alagoas em 2019 e atualmente estou no terceiro período.
- Tenho interesse em contribuir e aprender em projetos que envolvam Inteligência artificial e/ou ciência dos dados e/ou otimização e análise de algoritmos.

O que aprendi no curso

Revisão sistemática de literatura: passo a passo de como fazer uma revisão sistemática, elaboração e apresentação de um protocolo de uma revisão, entre outros.

Visualização de dados com python: como ler e manipular base de dados com o pandas, como responder perguntas a partir de uma base de dados, como plotar diferentes tipos de gráficos utilizando a matplotlib, entre outros.

O que aprendi no curso

Noções de inteligência artificial: uma ideia de machine learning, knn, deep learning, reinforcement learning, entre outros.

Cidades inteligentes e IoT: noções do que é IoT e suas aplicações, projetos no tinkercad envolvendo arduino uno, entre outros.

Agora sim, vamos falar sobre o projeto

Afinal, o que é o Linear Assignment Problem?

Formalmente, a instância do problema possui vários *agentes* e várias *tarefas*. Um agente X consegue executar um subconjunto do conjunto de todas as tarefas, sendo assim é necessário realizar o maior número de tarefas possível atribuindo no máximo um agente a cada tarefa e no máximo uma tarefa a cada agente.

Contextualizando...

Imagine o seguinte problema: Existem M candidatos que querem um emprego e N empregos disponíveis. Cada candidato tem um subconjunto de empregos nos quais tem interesse, e cada vaga de emprego só pode ser assumida por uma única pessoa. Sabendo disso, encontre uma atribuição de empregos aos candidatos de forma que o maior número possível de candidatos consiga algum emprego.

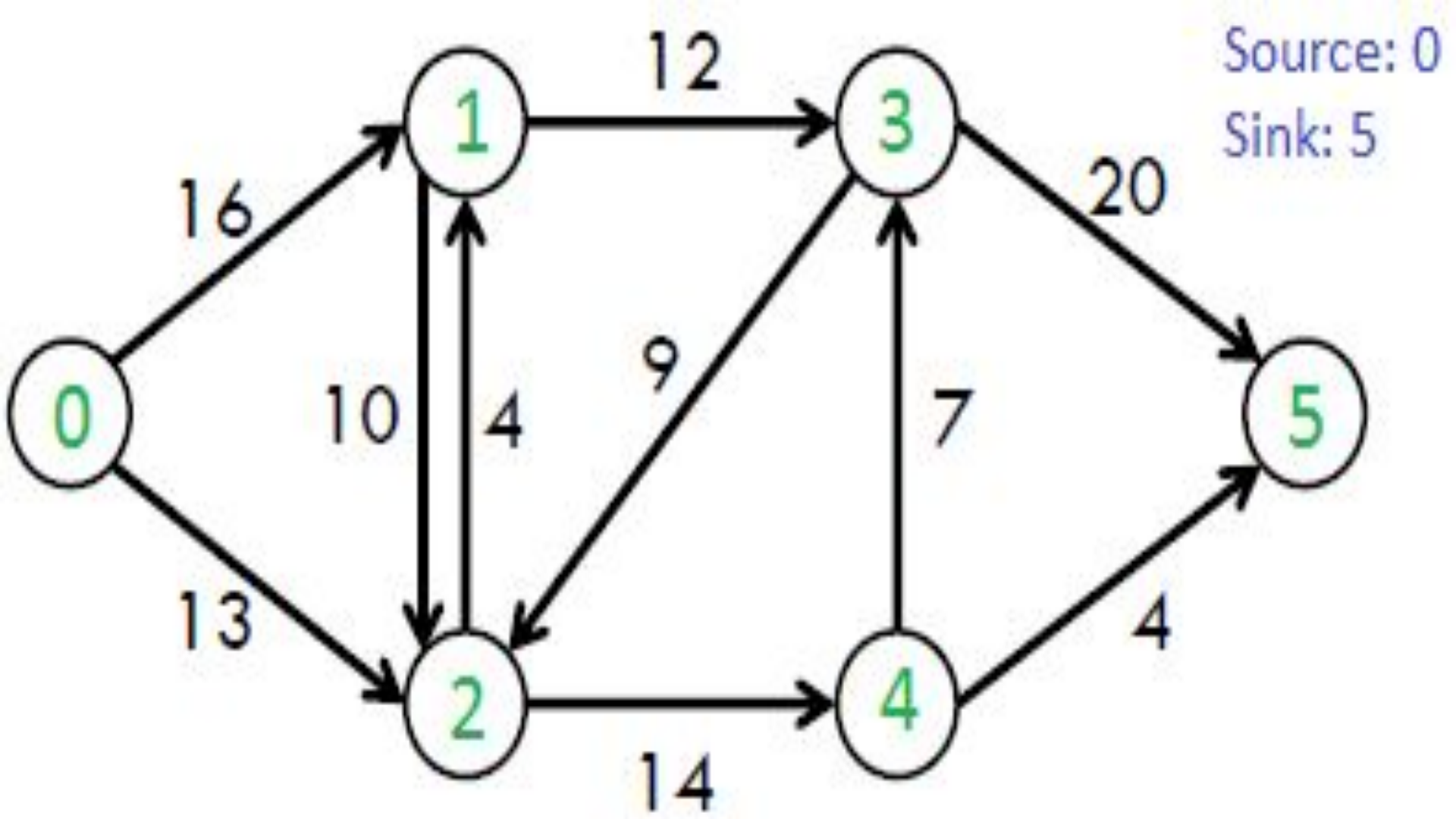
E como podemos solucionar isso?

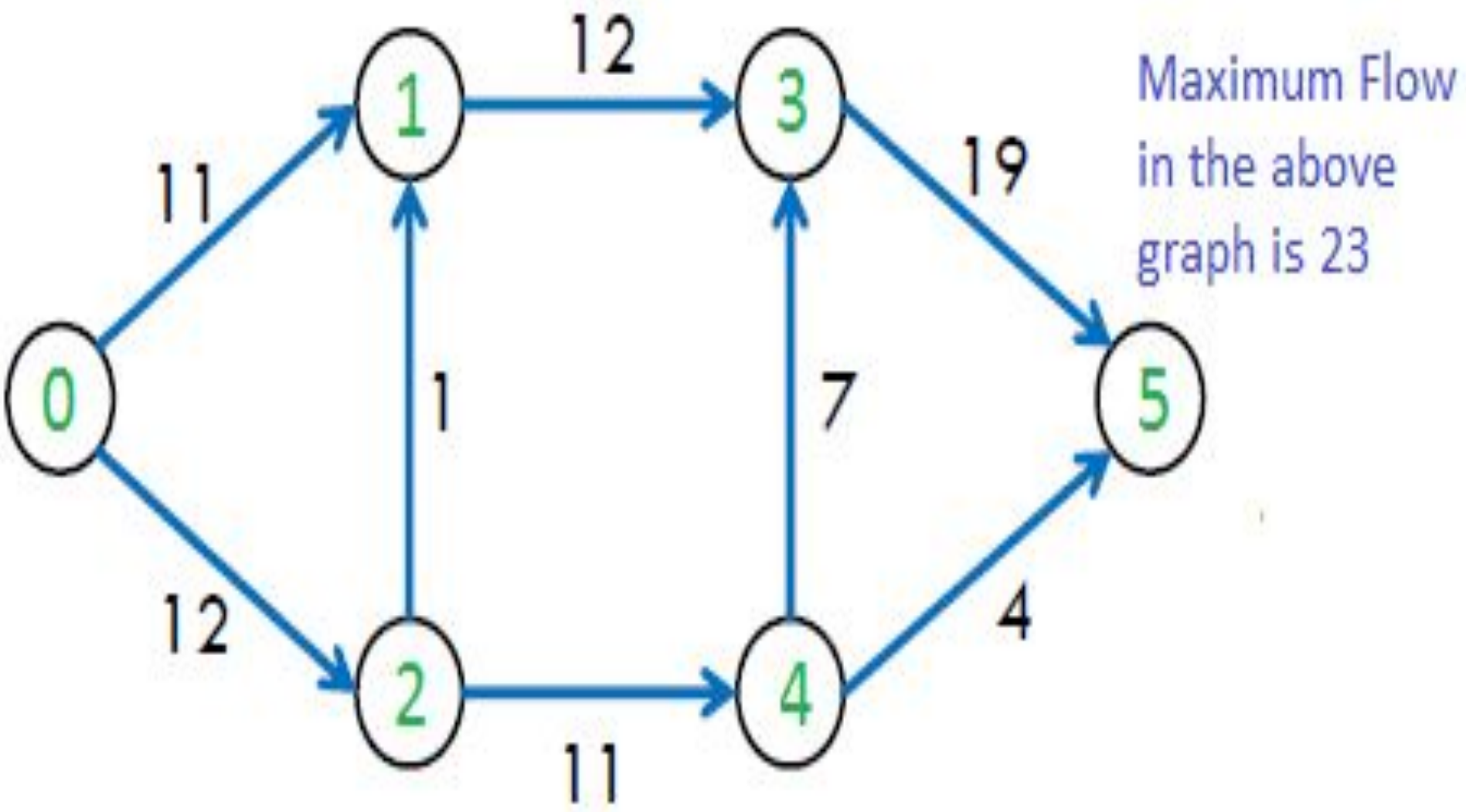
Esse problema, na verdade, pode ser modelado como um problema de fluxo máximo em um grafo!

Usaremos o algoritmo de Dinic para resolver esse problema.

E o que é o problema do fluxo máximo?

Imagine que temos uma rede de transporte de fluidos como um grafo conexo e direcionado G, no qual cada aresta possui uma capacidade C, nós queremos realizar um transporte de um líquido entre um vértice U até um vértice V, de modo que, nenhuma aresta receba mais do que a sua capacidade, sendo assim, qual a maior quantidade de líquido que podemos enviar de U para V ?


Source: 0
Sink: 5
0
1
2
3
4
5
16
13
12
10
4
9
7
20
4
14


12
1
3
11
19
Maximum Flow
in the above
graph is 23
0
1
7
5
12
4
2
4
11

Algoritmo de Dinic

O algoritmo pode ser descrito no seguinte passo a passo:

1) Vamos rodar um BFS (Breadth-First Search) partindo do vértice de início, com o objetivo de achar um “novo grafo”, de modo com que esse grafo possua apenas arestas que tenham uma capacidade restante maior do que zero.

Algoritmo de Dinic

O algoritmo pode ser descrito no seguinte passo a passo:

2) Caso o “novo grafo” exista, iremos rodar vários DFS (Depth-first search), de modo com que a prioridade sempre é chegar no vértice destino e retornar imediatamente após chegar nele, e em seguida, pegamos a aresta de menor capacidade e que está presente nesse caminho e subtraímos o valor dela da capacidade das demais arestas do caminho.

Algoritmo de Dinic

O algoritmo pode ser descrito no seguinte passo a passo:

3) Quando “novo grafo” não tiver caminho de U até V, retornaremos e daí temos o valor do fluxo máximo e consequentemente a resposta.

```cpp
// função principal do algoritimo
int dinic()
{
  int max_flow = 0;

  while (bfs() == true) // enquanto existir um "novo grafo" que possua caminho de source até sink
  {
    memset(start, 0, sizeof(start)); // start guarda a posição que em que devemos começar
                                     // em cada lista de adjacencia no dfs
    while (true)
    {
      int flow = dfs(source, INF); // o dfs me retorna o fluxo que deu para passar
                                   // em um novo caminho que ele achou de source até sink no "novo grafo"

      if (flow == 0) // se não for possivel passar mais fluxo por nesse "novo grafo", paramos a execução
      {
        break;
      }

      max_flow += flow; // caso contrário, iremos incrementar a resposta com o que encontramos
    }
  }

  return max_flow; // ao final teremos o fluxo máximo
}
```

E como iremos montar o grafo para resolver o LAP?

Sabendo disso, iremos montar a nossa rede de fluxo da seguinte maneira:

1) Criar dois vértices, um é o vértice de início e o outro vértice destino.

2) Para cada candidato, iremos criar um vértice equivalente a ele.

3) Para cada emprego disponível, iremos criar um vértice equivalente a ele.

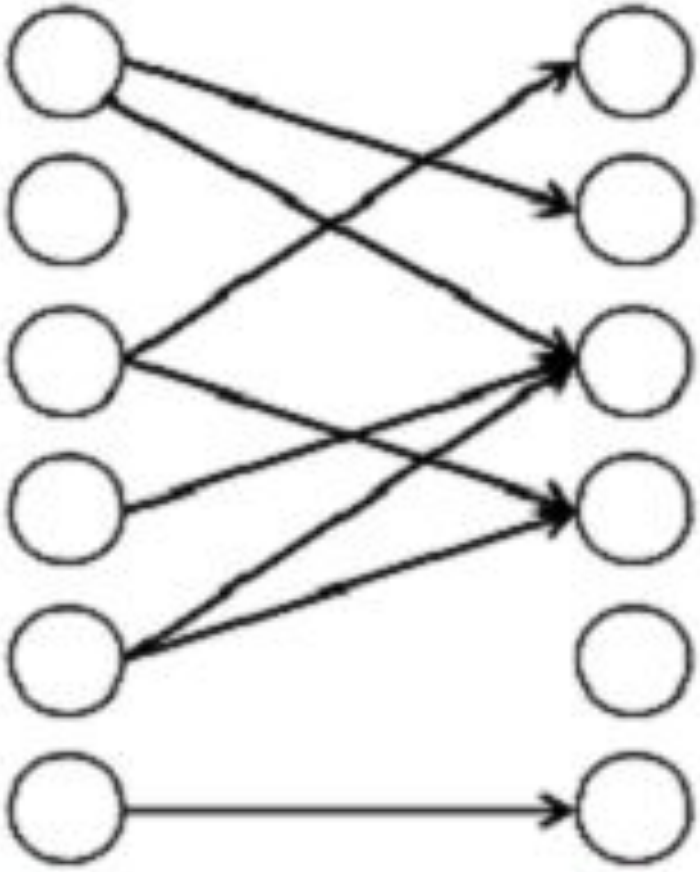
E como iremos montar o grafo para resolver o LAP?

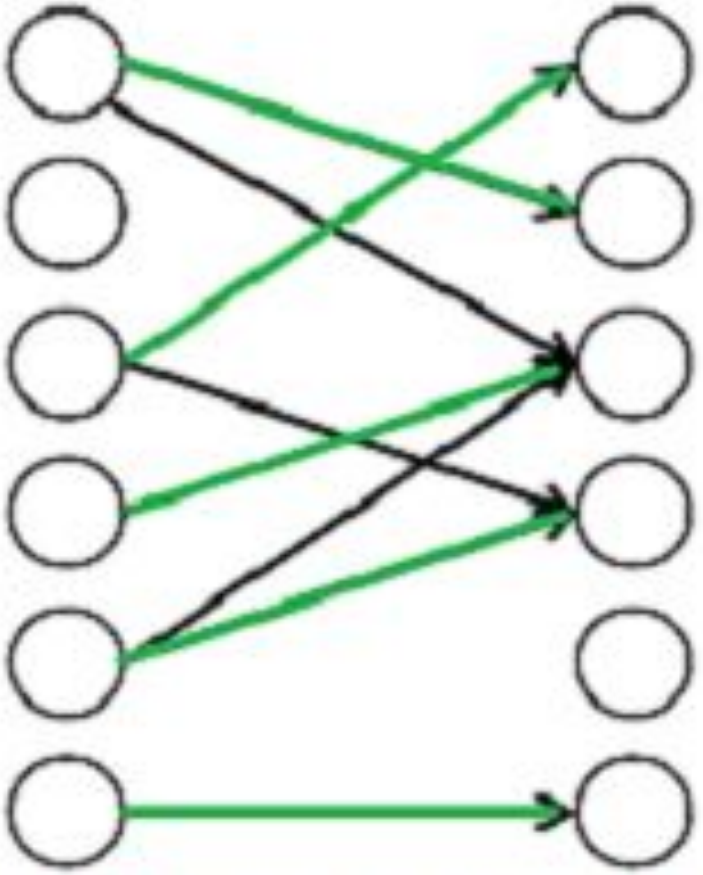
Sabendo disso, iremos montar a nossa rede de fluxo da seguinte maneira:

4)Com isso, para cada vértice que representa um candidato, adicionaremos uma aresta que vai do vértice de início até ele.

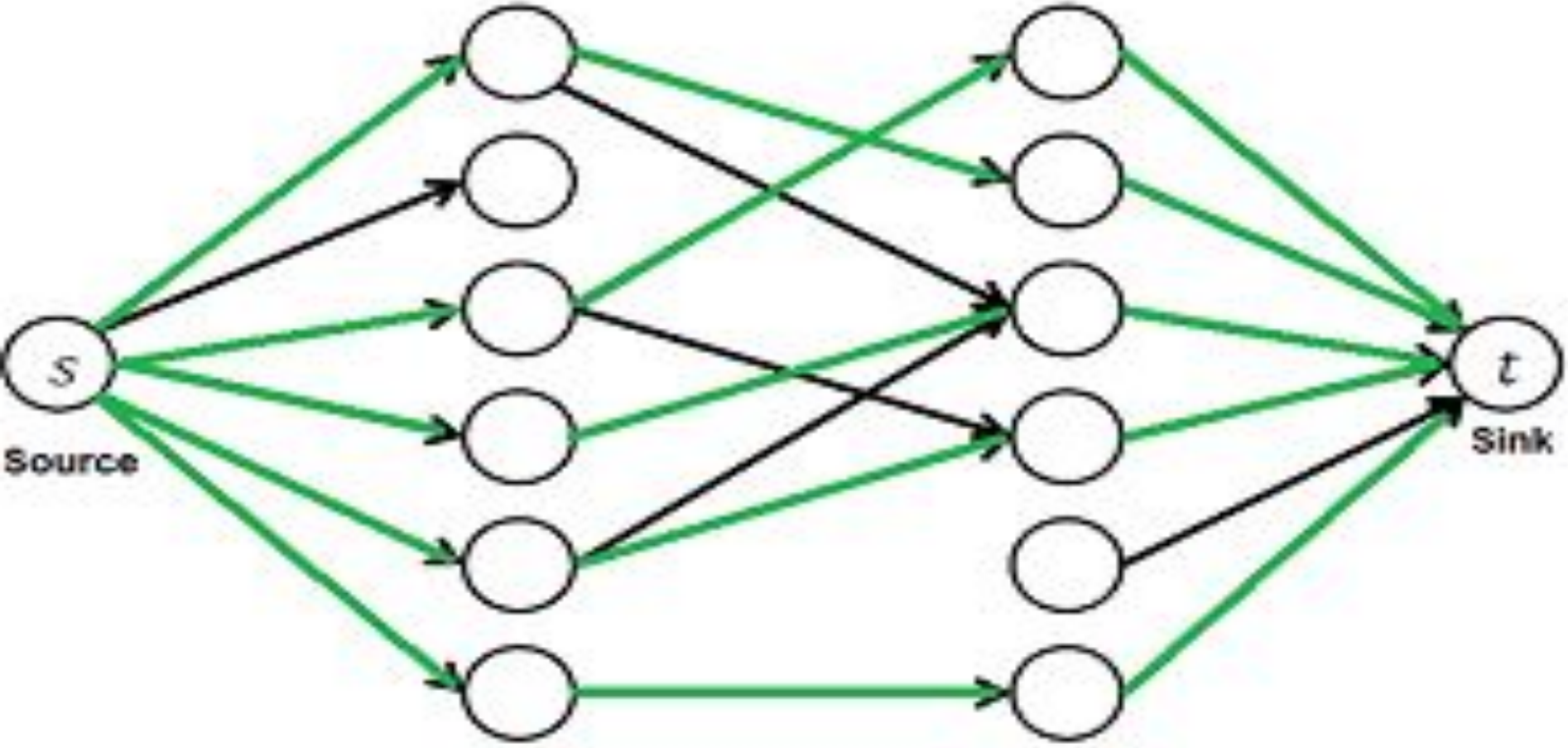
5) Para cada emprego Y que um candidato X esteja interessado, iremos criar uma aresta que vai do candidato X até o emprego Y.

6) Para cada emprego disponível X, adicionaremos uma aresta que vai de X até o vértice de destino.





Maximum five people can get jobs
(Maximum Matching)



The maximum flow from source to sink is five units. Therefore, maximum five people can get jobs.

```
joao@joao-Lenovo-ideapad-330-15IKB:~/Área de Trabalho/AssignmentProblemProject$ g++ main.cpp -o main_executable
joao@joao-Lenovo-ideapad-330-15IKB:~/Área de Trabalho/AssignmentProblemProject$ ./main_executable <test_case.in
4 candidates were selected!

the applicant 1 got the job 2
the applicant 3 got the job 3
the applicant 4 got the job 4
the applicant 2 got the job 6

nobody got the job 1
nobody got the job 5

joao@joao-Lenovo-ideapad-330-15IKB:~/Área de Trabalho/AssignmentProblemProject$
```

Com isso, chegamos ao fim :)

Meus agradecimentos a todos que de certa forma contribuíram para que esse curso de primavera pudesse acontecer.

Obrigado!

Repositório do projeto: https://github.com/jonh14lk/AssignmentProblemProject

Referências

1) https://www.geeksforgeeks.org/maximum-bipartite-matching/

2) https://www.geeksforgeeks.org/dinics-algorithm-maximum-flow/

3)https://cp-algorithms.com/graph/dinic.html

4)https://en.m.wikipedia.org/wiki/Assignment_problem

5)https://www.geeksforgeeks.org/max-flow-problem-introduction/