Trabalho Prático 2 - Desemprego

Filipe de Araújo Mendes - 2021031920

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

flipeara@ufmg.br

1. Introdução

O trabalho prático 2 propõe um problema em encontrar a quantidade máxima de pares que podem ser formados, dado um grafo bipartido, ou seja, o problema do emparelhamento máximo. É pedido que se resolva o problema de duas maneiras diferentes, uma com um algoritmo guloso, que não necessariamente produzirá o resultado ótimo, e outra com um algoritmo exato visto em sala, que deve apresentar o resultado certo.

2. Modelagem

O algoritmo teria que trabalhar com dois tipos diferentes de dados distintos, as pessoas e os empregos. Como existe uma ligação entre uma pessoa e um emprego, é possível criar um grafo, no qual os vértices são as pessoas e os empregos, as arestas são a possibilidade de uma pessoa conseguir o emprego e, por fim, o grafo será bipartido já que pessoas só se relacionam com empregos e vice-versa.

Para as pessoas e empregos, que seriam lidos em uma string, foi utilizado um unordered_map, para traduzir as strings para números, que seriam utilizados como vértices no grafo.

2.1. Grafo

Tendo em mente que o algoritmo teria que percorrer os vértices adjacentes a outro vértice, assim como a não garantia de que os grafos teriam n^2 arestas, escolhi fazer o grafo com uma lista de adjacência, já que facilitaria para encontrar os vizinhos de um vértice e não teria espaço desperdiçado, assim como a matriz teria, por não existir muitas arestas incidentes a um vértice.

Já que o grafo tem tamanho fixo, conhecido no início da execução do programa, foi decidido que os vértices seriam representados em um vector, que possui uma forma eficiente de acesso aos elementos, podendo utilizar vector[i] para acessar a posição desejada.

Os vértices em si, guardam a quantidade de arestas incidentes a ele, além de guardar as arestas.

As arestas, por sua vez, teriam que ser lidas uma por uma e adicionadas ao grafo, portanto, vector não foi visto com a melhor opção, pela necessidade de realocação sempre que ele atinge o tamanho máximo, assim, foi escolhida a estrutura list para representar as arestas adjacentes a um vértice. Por fim, a aresta precisava de tres coisas: o outro vértice, o fluxo entre os dois vértices e um ponteiro para a aresta de retorno.

2.2. Algoritmo guloso:

Para resolver o problema de forma gulosa, foi decidida a seguinte solução: dado um grafo bipartido, com pessoas, empregos e arestas entre eles, o algoritmo seleciona a pessoa que tem a menor quantidade de empregos disponíveis e, dos empregos disponíveis para ela, escolhe o emprego que possui a menor quantidade de pessoas concorrendo à vaga. Então, o algoritmo marca os dois como visitados, já que eles não poderão fazer parte de um segundo par, e atualiza a quantidade de vagas para as pessoas que estavam concorrendo àquele emprego.

Por exemplo:

Suponha que todas as pessoas possuem 3 oportunidades de emprego, exceto o José, que possui somente 2: “caixa de supermercado” e “atendente de call center”. Dessas duas vagas, “caixa de supermercado” possui 4 pessoas concorrendo a vaga e “atendente de call center”, 3. Assim, o algoritmo escolherá o par (José, ”atendente de call center”), já que os dois possuem a menor quantidade de arestas, marcará os dois como visitados e, reduzirá em um o número de vagas para todas as pessoas que estavam concorrendo para “atendente de call center”.

Para a análise de complexidade do algoritmo, vejamos um pseudocódigo:

1.	Marcar todos os vértices como não visitados	O(n)
2.	M vezes	O(m)
2.	Encontra a pessoa com menor quantidade de ofertas	O(p)
3.	Encontra o emprego com menor quantidade de candidatos	O(e)
4.	Marca os dois como visitados	O(1)
5.	Reduzir vagas para candidatos do emprego	O(p)

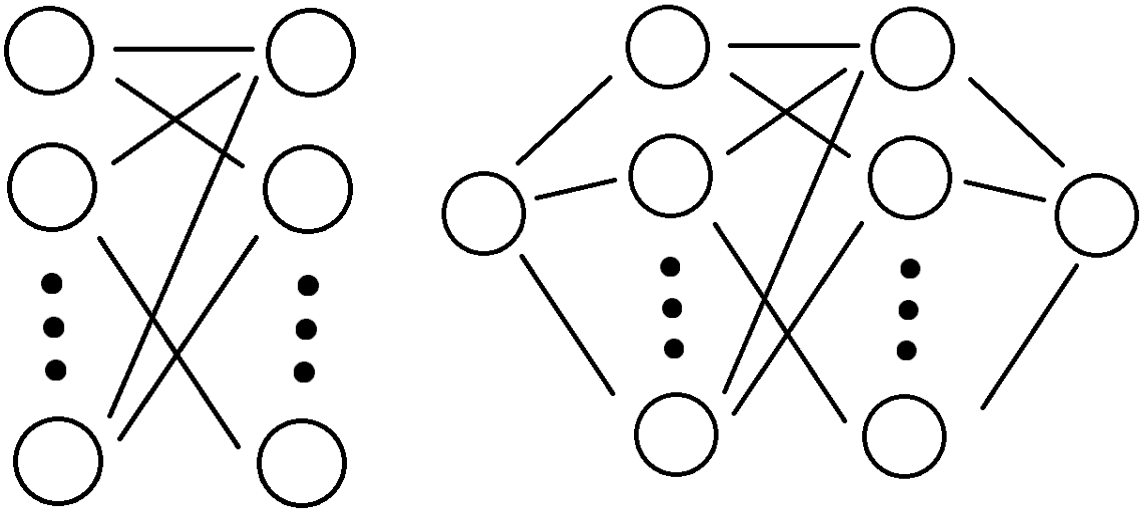
É importante notar que p + e = n, então o algoritmo fica com uma complexidade de:

O(n) + O(m) * (O(p) + O(e) + O(1) + O(p) =
O(n) + O(m) * O(n) =
O(m * n)

2.2. Algoritmo exato:

Um algoritmo que consegue produzir o resultado correto do problema é o Ford-Fulkerson, que acha o fluxo máximo de um grafo, dado um “Source” e um “Sink”. Para isso, o grafo original, bipartido com arestas de pessoas para empregos, precisa ser alterado, adicionando dois vértices: um “Source”, que possui uma aresta para cada pessoa, e um “Sink”, que possui uma aresta de cada emprego para ele.

Assim, temos um grafo pronto para utilizar um algoritmo para achar o fluxo máximo, considerando que cada aresta possui fluxo máximo 1.



Grafo original e o grafo alterado.

A complexidade do Ford-Fulkerson é O(m * f), sendo “e” o número de arestas do grafo e “f” o fluxo máximo do grafo.

3. Comparação

O algoritmo guloso utilizado, nos casos de teste, conseguiu entregar o resultado correto para o número máximo de pares que podem ser formados, mas não foi realizado um estudo para saber se ele, de fato, produz em todos os casos o resultado correto. Em questão de tempo, pelos testes, ele executou consideravelmente mais rápido que o exato. Para exemplificar, foi testado o caso 13, que executou em 0,1 segundo com o guloso, mas levou 0,7 para o exato. Considerando que está incluso nesse tempo a criação do grafo, a diferença entre o exato e o guloso pode ser ainda maior do que aparenta ser.

Tendo isso em vista, o algoritmo guloso é recomendado em casos que aceitam resultados não necessariamente 100% exatos e que quer ser executado de forma mais rápida. O exato, apesar de demorar mais, possui uma prova de que ele entrega o resultado 100% em todos os casos, portanto é recomendado em casos que não pode ocorrer nenhum erro, querendo realmente maximizar o emparelhamento.