Exercícios de Parametrizada

Matheus Souza D'Andrea Alves

2018.2

Bounded search tree

Cluster editing

Observe as seguintes definições:

Definição 1: Grafo Cluster

Um grafo G onde toda componente conexa δ_i(G) é uma clique.

Definição 2: Grafo livre de P_3

Um grafo G é chamado de livre de P_3 se e somente se não possuir um grafo caminho induzido com 3 vértices.

Dessa forma demonstraremos o seguinte teorema.

Teorema 1: Um grafo G é cluster se e somente se é livre de P_3.

Demonstração.
Suponha por absurdo que G não é livre de P_3 isso implica em que exista um subgrafo induzido G' onde existem dois vértices não vizinhos na mesma componente, isso implica que alguma componente de G não é uma clique o que é absurdo.

Suponha agora que um grafo H qualquer não possua P_3 isso implica em que dado quaisquer pares de vértices $u, v \in V(G)$ ou $\exists(u, v) \in E(G)$, ou os vértices estão em componentes distintas de H, e portanto H é um cluster. □

Abordaremos o problema de cluster editing para esse exercício conforme descrito abaixo.

Problema 1: Cluster editing

Entrada: *Um grafo G um inteiro k*
Questão: É possível transformar G em um grafo cluster, adicionando/removendo k arestas?

Como vimos acima, um grafo cluster é livre de P_3 e portanto podemos abordar o problema de cluster editing como o problema de eliminação de P_3 em um grafo G. Para eliminar um p3 existem 3 possibilidades:

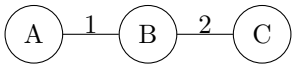

Figura 1: Um P3

- Remover aresta 1
- Remover aresta 2
- Adicionar aresta entre A e C

Sabemos que é possível encontrar P_3 em tempo $\mathcal{O}(n+m)$ portanto usando o número de movimentos restantes como paramêtro segue o algoritmo.

```
1  Function ClusterEdit(Graph g, int remainingMovements): Graph{
2    if g.hasP3()  {
3      if remainingMovements = 0 {
4        return null
5      } else {
6        var p3 = g.getP3()
7        var g1 = g.removeEdge(p3.edges[0])
8        var g2 = g.removeEdge(p3.edges[1])
9        var g3 = g.addEdge(p3.vertices[0],p3.vertices[2])
10       remainingMovements--
11       return ClusterEditable(g1,remainingMovements)
12             or ClusterEditable(g2,remainingMovements)
13             or ClusterEditable(g3,remainingMovements)
14     }
15   } else
16       return g
17 }
```

Tal algoritmo é resolvível em tempo $\mathcal{O}(k^3(n+m))$

Cadeia mais próxima

Observe as seguintes definições

Definição 3: Distancia entre strings

Um inteiro que representa quantos caracteres são diferentes entre uma string s e uma string s'

Problema 2: Cadeia mais próxima

Entrada: *Um conjunto de strings S um inteiro d.*
Questão: Existe uma string s tal que $distance(s, s_i) \leq d; \quad \forall s_i \in S$

Um parâmetro intuitivo para o problema é a distância.

Usando esse parâmetro podemos formular o seguinte algoritmo, suponha o seguinte conjunto S de strings:

ADB	AAB
BDB	ABD
JBD	QAB

Escolha uma string $s \in S$. Observe que para cada string em S temos uma distânca a atual string.

Suponha $s = ADB$ e $d = 2$, nossas distâncias são:

0	1
2	2
3	2

Escolha agora um $s' \in S$ qualquer tal que $distance(s, s') > d$. Portanto seja $s' = JBD$, escolhemos as $d + 1$ posições onde s difere de s'

```
0 1 2
A D B
J B D ⟹ {0, 1, 2}
J B D
```

Para cada posição diferente realize a troca do caracter na string atual, logo:

JDB:

```
1 2
1 3
2 2
```

ABB:

1	1
1	1
2	2

ADD:

1	2
1	1
2	3

Cada instância gerada deve estar a uma distância $d' = d - 1$ para a próxima iteração. Observe que, se $distance(s, s_i) > d + d'$ esta instância é inviável, pois seria impossível reduzir a distânca para o esperado com os movimentos possíveis.

Observe que em nosso exemplo já encontramos ABB que é uma solução possível, pois $\forall s \in S\ distance(s, ABB) \leq 2$.

O algoritmo é sintetizado a seguir.

```
Function ClosestString(Set<string> set,int maxDistance,
                       string current, int remainingDistance): string {
    if remainingDistance < 0 {
      return null
    }
    if set.Any(str -> {distance(current,str) > maxDistance + remainingDistance}){
      return null
    }
    if set.All(str -> {distance(current,str) <= maxDistance }){
      return current
    }
    var distant = set.First(str -> {distance(str,current) > maxDistance})
    var differentLettersPositions = distant.PositionOf(pos -> {
      distant[pos] != current[pos]
    })
    var range = differentLettersPositions[0...maxDistance]
    for position in range do {
      var newCurrent = current
      newCurrent[position] = distant[position]
      var result = ClosestString(set,maxDistance,newCurrent,remainingDistance-1)
      if result not null {
        return result
```

```
23          }
24        }
25        return null
26    }
```