PAA - Aula 16
Union Find, k-Clusterização com Espalhamento Máximo

Union Find é uma estrutura de dados especializada em manter
    - o registro das partes (subconjuntos) em que se divide um conjunto.
- Na última aula usamos essa estrutura no algoritmo de Kruskal,
    - para consultar a que componente conexa pertence cada vértice.

Essa estrutura suporta três operações:
- makeSet(e) - cria uma parte (subconjunto) apenas com o elemento ‘e’
- find(e) - devolve o representante da parte em que está ‘e’
- union(e, f) - une a parte de ‘e’ com a parte de ‘f’

Em uma das implementações mais simples dessa estrutura,
- todo elemento de uma parte aponta para o representante da mesma.
    - Com isso, as operações makeSet e find levam tempo O(1).
- No entanto, a operação union pode levar tempo O(n),
    - o que acontece se os conjuntos envolvidos tem muitos elementos.

Curiosamente, um pequeno detalhe da implementação
- pode tornar essa implementação eficiente.

Para tornar nossa implementação do Union Find mais eficiente, em uma união
    - apenas o representante da menor parte envolvida é alterado.
- Note que, mesmo nesse caso uma operação union pode levar tempo O(n),
    - caso cada conjunto tenha n/2 elementos, por exemplo.
- No entanto, o custo amortizado da union passa a ser log(n), i.e.,
    - o custo médio das operações union é O(log n).

Para verificar isso, foque no representante de um elemento ‘e’.
- Cada vez que esse representante é alterado, quer dizer que após uma união
    - ‘e’ é parte de um conjunto com pelo menos o dobro de elementos.
- Sendo k o número de alterações que o representante de ‘e’ sofreu,
    - temos que o número de elementos no conjunto de ‘e’ <= 2^k <= n.
- Portanto, k <= log n. Somando ao longo dos n vértices,
    - o total de alterações de representante causadas por unions <= n log n.

Essa implementação já é suficiente garantir que o algoritmo de Kruskal,
    - que vimos na última aula, rode em tempo O(m log n).
- No entanto, existem melhorias na implementação do Union Find,
    - que levam ele a gastar tempo O(log n) por operação,
        - e até mesmo tempo constante na prática.
- Dessas melhorias de implementação, destacamos:
    - Lazy unions, Union by rank, e Path compression.

Problema da k-clusterização com espalhamento máximo:

Entrada:

- um conjunto de pontos S,
- uma função d( ) de distância entre os pontos,
    - indicando quão diferentes são dois elementos de S,
- e um inteiro positivo k indicando o número de clusters desejado.

Solução:

- agrupar os pontos em k clusters,
    - de modo a maximizar a menor distância entre clusters.
- A distância entre dois clusters A e B é dada
    - pela distância entre os pontos mais próximos,
        - i.e., d(A, B) = min_{p \in A, q \in B} d(p, q).
- Assim, o objetivo é encontrar uma solução que maximize
    - min_{p e q separados} d(p, q),
- sendo que dois pontos p e q estão separados
    - se pertencem a clusters distintos.

Estratégia gulosa:

A função objetivo é a menor distância entre qualquer par de pontos separados,

- i.e., que não pertencem ao mesmo cluster.

Por isso, nossa estratégia gulosa é unir, em cada iteração,

- o par de pontos separados mais próximo,
    - o que deve aumentar o valor da função objetivo.

Observe que cada união funde dois clusters. Como começamos

- com cada um dos ‘n’ pontos isolados e queremos terminar com k clusters,
    - o algoritmo termina depois de n - k iterações/fusões.

Note que este algoritmo é uma variante do algoritmo de Kruskal, em que

- pontos são vértices,
- pares de pontos são arestas,
- distância entre pares são custos das arestas,

e que para quando obtém k componentes conexas,

- que são os clusters.

Prova de corretude da estratégia:

Seja C_1, …, C_k uma clusterização gulosa

- com valor de espaçamento (função objetivo) L

Seja C'_1, …, C'_k uma clusterização qualquer com valor de espaçamento L'.

Vamos mostrar que L' <= L.

- Note que, como C'_1, …, C'_k é uma solução arbitrária,
    - isso implica a otimalidade de C_1, …, C_k.

Caso 1: Se C_1, …, C_k e C'_1, …, C'_k são iguais, então

- elas tem o mesmo espaçamento e o resultado segue.

Caso 2: Se as soluções são diferentes, seja 'p' e 'q' um par

- que está no mesmo cluster na solução gulosa (C_1, …, C_k)
    - e em clusters diferentes na outra solução (C'_1, …, C'_k).
- Note que um par assim deve existir, já que ao menos um elemento
    - deve ter vizinhos diferentes nas duas clusterizações.

Agora temos um subcaso fácil e um difícil.

Subcaso fácil: ‘p’ e ‘q’ foram unidos diretamente pelo algoritmo guloso.

- Como o algoritmo guloso une sempre o par separado mais próximo,
    - neste caso o espaçamento (L) da solução gulosa é pelo menos d(p, q),
- i.e., L >= d(p, q), pois para chegar a unir ‘p’ e ‘q’
    - o algoritmo guloso já uniu todos os pares mais próximos que ‘p’ e ‘q’.

Mas, como ‘p’ e ‘q’ estão em clusters diferentes na solução (C’_1, …, C’_k),

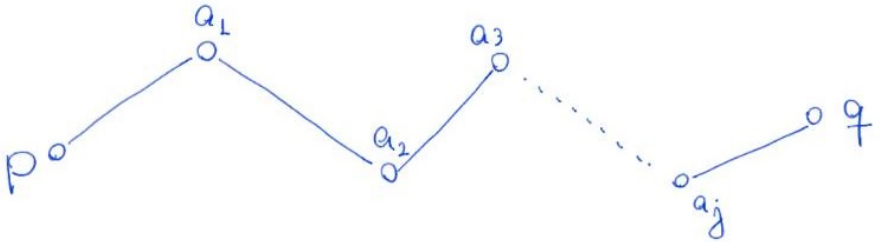
- temos que o espaçamento dela é no máximo L, i.e., L’ <= d(p, q) <= L.

Subcaso difícil: ‘p’ e ‘q’ não foram fundidos diretamente pelo algoritmo guloso,

- i.e., eles acabaram no mesmo cluster porque algum elemento do cluster
    - de ‘p’ foi fundido com algum elemento do cluster de ‘q’.

Observe que, como todo elemento começa isolado, podemos descrever

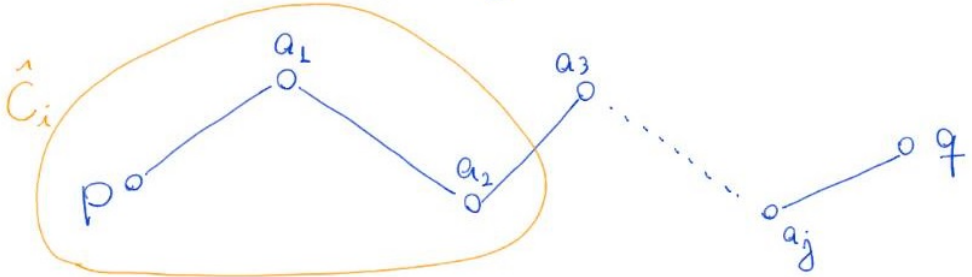
- um caminho das fusões que levaram ‘p’ e ‘q’ a terminarem juntos.



    - No desenho, cada aresta indica uma fusão direta.

Seja C'_i o cluster da outra solução em que está 'p'. Observe que,

- como 'p' e 'q' estão em clusters distintos em C'_1, …, C'_k,
- em algum ponto do caminho que começa em 'p' e termina em 'q'
  - devemos sair do cluster C'_i.



No nosso exemplo, observe que o par (a_2, a_3) está no mesmo cluster

- na solução gulosa e está em clusters distintos em C'_1, …, C'_k.
- Em geral, chamemos de (a_r, a_s) o primeiro par do caminho
  - que atravessa a fronteira do cluster C'_i.

Para concluir a prova basta observar que

- (a_r, a_s) foram fundidos diretamente na solução gulosa.
  - Portanto, d(a_r, a_s) <= L.
- (a_r, a_s) estão em clusters distintos em C'_1, …, C'_k.
  - Portanto, L' <= d(a_r, a_s).
- Assim, L' <= d(a_r, a_s) <= L.