Divisão e Conquista

- É uma técnica que usar recursão e por isso é descendente
- **Ideia:**
  1. **Dividir** o problema em um ou mais subproblemas
  2. **Conquistar** os subproblemas, resolvendo-os recursivamente
     Daí, problema pequeno o bastante -> resolve direto
  3. **Combinar** as soluções encontradas dos subproblemas, a fim de formar uma solução para o problema original
- Exemplo: obter os valores máximo e o mínimo de um vetor
- Desvantagem: Se o **balanceamento** na subdivisão de um problema for ruim, a complexidade é ruim -> dá no pior caso . Um exemplo disso é o quicksort → **Escolhe o pivô**

Estratégia Gulosa

- São tipicamente usados para resolver problemas de **otimização**
- **Ideia:** Quando temos uma escolha a fazer, fazemos aquela que pareça ser a melhor no momento (melhor local) na esperança de obter uma solução ótima global.
  [Essas escolhas são irreversíveis]
- Nem sempre dão a solução ótima, mas quando dão são muito rápidos
- Exemplos: árvore geradora mínima, caminho mínimo a partir de uma fonte, compressão/codificação de dados
- Em cada passo, a escolha deve ser feita com as seguintes características:
  - **Possível:** deve satisfazer as restrições do problema
  - **Localmente ótima:** A melhor entre todas disponíveis naquele momento
  - **Irreversível:** uma vez feita, não pode ser alterada
- Aplicação: Caminho mínimo com uma fonte

🔎 Definição:

> Dado um grafo ponderado G = (V,E), desejamos obter o caminho mais curto (de menor peso) a partir de uma dada fonte até qualquer outro vétice pertencente a V.

- Para esse problema, usamos o algoritmo de Dijkstra **(usa estratégia gulosa e (pesos não negativos)**

→ **Como ele funciona ???**

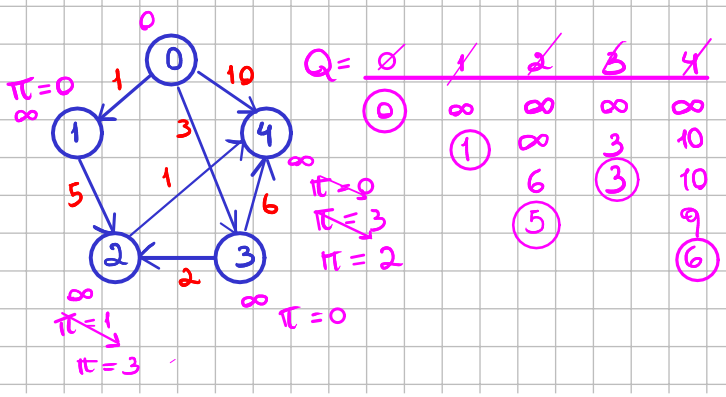
```
Relax (u,v,w)
// w: peso da aresta (u,v)
1. se (d[v] > d[u] + w(u,v)) então
2.    d[v] = d[u] + w(u,v)
3.    p[v] = u
```

← antecessor

```
Dijkstra (G,w,s)
1.   para cada u em V[G]
2.      d[u] = infinito
3.      p[u] = nil
4.   d[s] = 0
5.   S = { }
6.   Q = nova fila de prioridades em V[G]
7.   enquanto Q não for vazia
8.      u = EXTRACT-MIN(Q)
9.      S = S + u
10.     para cada v em Adj[u]
11.        Relax (u,v,w)
12.  retorne (p)
```

inicializando

← Aqui está a estratégia gulosa **(não tem volta essa escolha)**



Q =

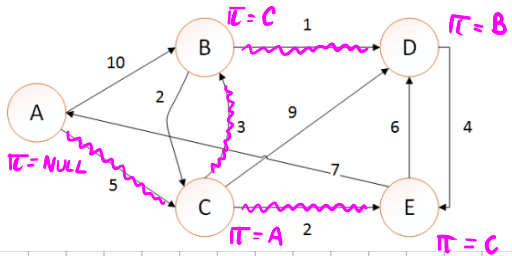
	0	1	2	3	4
0	∞	∞	∞	∞	
1		∞	∞	3	10
3			6	3	10
2			5		9
4					6

Estruturas para implementar a fila

	Árvore Balanceada	Vetor Ordenado	Heap	Vetor Não Ordenado
EXTRACT-MIN	logn	O(1)	logn	n
ATUALIZAR ESTIMATIVA	logn	n	logn	1
	V.logV + E. logV = O(E.logV)	V + E.V = O(E.V)	V.logV + E.logV = O(E.logV)	V² + E = O(V²)

pq eu preciso reordenar ao atualizar a estimativa (na vdd a complex. depende do alg. de ordenação)

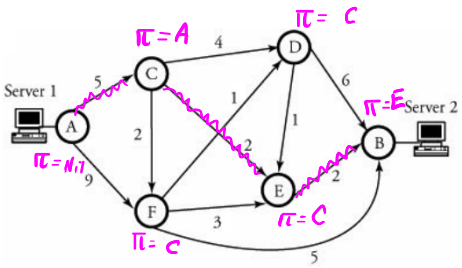
- Use o algoritmo de Dijkstra para encontrar o caminho mínimo entre o vértice *A* e os demais vértices do grafo abaixo.



Q =

	A	B	C	D	E
0	∞	∞	∞	∞	
10		5	∞	∞	
6			14	7	
13					
9					

- O grafo abaixo retrata o esquema de uma rede de comutação com 6 (seis) nós e os respectivos custos das arestas que ligam esses nós. Encontre o caminho de menor custo entre o Server 1 e o Server 2 usando o algoritmo de Dijkstra.



Q =

	A	B	C	D	E	F
0	∞	∞	∞	∞	∞	
9		5	9	7	9	
7						

S = {A, C, E, F, B, D}

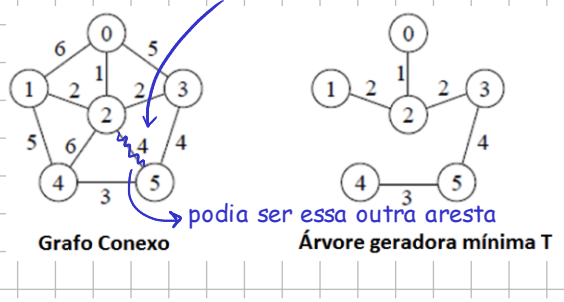
Árvore geradora mínima (AGM)

🔎 **Definição:**

> AGM é um grafo acíclico conexo que contém todos os vértices do grafo. O problema da AGM consiste em encontrar uma AGM dado um grafo ponderado conexo

- Exemplo:

[OBS.: Essa árv. não é única]



- Algoritmo de Prim

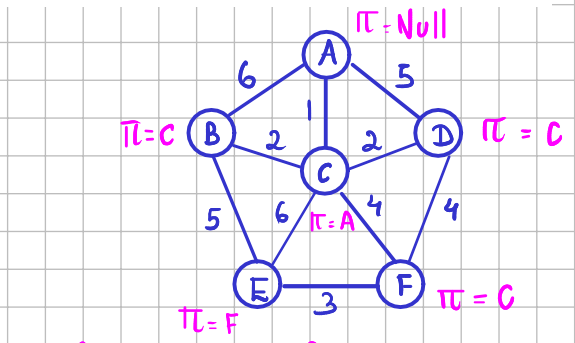
✔Sempre leva a uma AGM

✔Passo 1: Escolher um vértice arbitrário do conjunto V

✔Passo 2: Em cada iteração, expandimos a árv. corrente de maneira **gulosa** (não tem volta), juntando a ela o **vértice mais próximo** (com conexão de menor peso) que ainda não pertence a árvore.

✔É basicamente um Dijkstra sem considerar o acúmulo da estimativa anterior

```
AGM-PRIM (G,w,s)
1.    para cada u em V[G]
2.       d[u] = infinito
3.       p[u] = nil
4.    d[s] = 0
5.    S = { }
6.    Q = nova fila de prioridades em V[G]
7.    enquanto Q não for vazia
8.       u = EXTRACT-MIN(Q)        <-- A gula tá aqui
9.       S = S + u
10.      para cada v em Adj[u]
11.         se v pertence a Q e w(u,v) < d[v]
12.            d[v] = w(u,v)
13.            p[v] = u
14.   retorne (p)
```

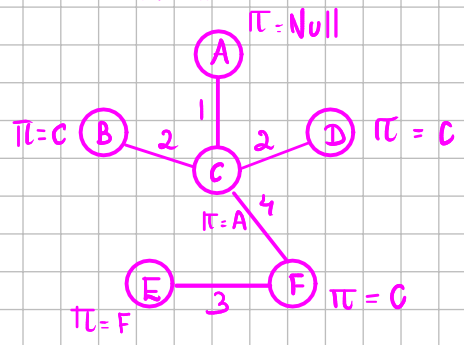


S = {A C B D F E}

Q =

A	B	C	D	E	F
0	∞	∞	∞	∞	∞
	6	1	5	6	4
	2		2	5	
				3	

Árvore Geradora Mínima:



✔A complexidade segue aquele mesmo esquema do Dijkstra e depende de qual estrutura foi usada para implementar a fila de prioridades

- Algoritmo de Kruskal

✔Também é guloso

✔Temos um conjunto A, que é uma floresta e toda aresta adicionada a A é uma **aresta de menor peso** que une duas árvores distintas

✔O processo que une 2 árv. é reptido até que exista apenas uma árvore na floresta

✔Usa a estrutura de dados para conjuntos disjuntos com as operações: criar um novo conjunto, unir dois conjuntos e encontrar o conjunto que contém um dado elemento

```
AGM-KRUSKAL(G, w)
1  A ← ∅
2  para cada v ∈ V[G] faça
3      MAKE-SET(v)
4  Ordene as arestas em ordem não-decrescente de peso
5  para cada (u, v) ∈ E nessa ordem faça
6      se FIND-SET(u) ≠ FIND-SET(v)
7          então A ← A ∪ {(u, v)}
8                UNION(u, v)
9  devolva A
```

Complexidade:

- Ordenação: $O(E \lg E)$
- $|V|$ chamadas a MAKE-SET
- $|E| + |V| - 1 = O(E)$ chamadas a UNION e FIND-SET

✔**É um algoritmo ótimo**

✔Basicamente, eu crio o conjunto A (inicialmente ele é composto pelos vértices)

✔Ordenas as arestas em ordem crescente

✔Itero pelas arestas e verifico se os conjuntos comparados são disjuntos. Se for, une eles.

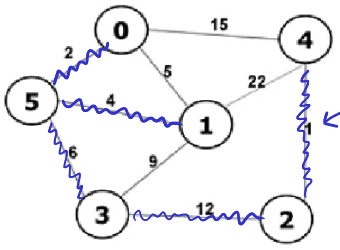
✔A complexidade é O(ElogE) considerando um Heap. Considerando um Heap no Prim também, podemos dizer que eles tem a mesma complexidade, assintoticamente. Porém, Depende do alg. de ordena.

Prim : O(ElogV) — Depende muito de V — Mais usado para grafos esparsos

Kruskal: O(ElogE) — Depende muito de E — Mais usado para grafos densos

- A prefeitura de uma cidade precisa pavimentar algumas ruas para que se possa ir de qualquer bairro para qualquer bairro só por rua pavimentada. No entanto, o prefeito quer minimizar ao máximo o total de quilômetros a pavimentar.

  Dado o mapa abaixo, mostre o conjunto de ruas a pavimentar que preencha todos esses requisitos.



Esse é um problema de árvore geradora mínima. Vamos de Kruskal: