MAC0323 - ALGORITMOS E ESTRUTURAS DE DADOS II

GRAFOS

DFS: busca em profundidade

- visita todos os vértices do grafo
- recursivo
- pilha

RDFS: retorna se tem caminho de u a todos

Rdfs(u, marked)

RBFS: retorna distâncias de u a todos

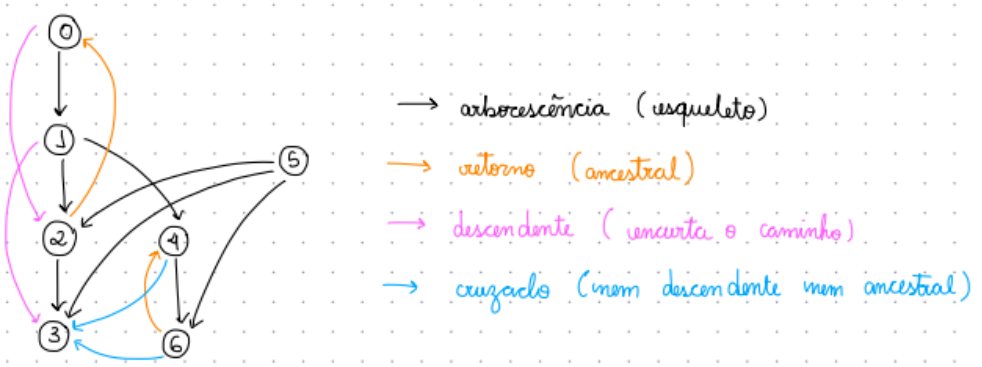
Rbfs(u, dist)

BFS: busca em largura

- o algoritmo começa por um vértice, digamos s. O algoritmo visita s, depois visita todos os vizinhos de s, depois todos os vizinhos dos vizinhos e assim por diante.
- visita apenas os vértices que estão ao alcance do vértice inicial.
- iterativo
- fila

TIPOS DE ARCOS

- **ARBORESCÊNCIA:** (esqueleto) os arcos da arborescência são usados na recursão da dfs.
- **RETORNO:** (ancestral) o arco u-v é chamado de retorno se v foi visitado antes de u e é ancestral dele.
- **DESCENDENTE:** (encurta o caminho) o arco u-v é descendente se v foi visitado depois de u e é descendente dele.
- **CRUZADO:** (nem descendente nem ancestral) o arco u-v é cruzado se v foi visitado antes do u e não é descendente nem ancestral.



ALGORITMO DE DIJKSTRA

- sabemos que a busca em largura nos dá o menor caminho entre um vértice inicial e todos os demais no grafo.
- o caminho é medido em número de arestas, ignorando quaisquer pesos que estas tenham.
- para vários problemas, contudo, o peso nas arestas é crucial, como quando queremos achar a rota mais curta entre duas cidades.

o que fazer nesses casos? Usamos o **algoritmo de Dijkstra:**

- algoritmo que calcula o caminho mais curto, em termos do peso total das arestas, entre um nó inicial e todos os demais nós no grafo.
- o peso total das arestas é a soma dos pesos das arestas que compõem o caminho.
- para cada vértice v do grafo, mantemos um atributo $d[v]$ que é um limite superior para o peso do caminho mais curto do nó inicial s a v.
- dizemos que $d[v]$ é uma estimativa de caminho mais curto, inicialmente feito ∞.
- Também armazenamos o vértice que precede v ($p[v]$ - precedente de v) no caminho mais curto de s a v.
- faça a estimativa de distância de s a qualquer vértice ser infinita.
  - exceto, claro, a distância de s a s, que é 0, ou seja, $d[s] = 0$ e $d[v] = \infty$ para todo $v \neq s$.
- faça os precedentes dos nós serem um valor qualquer
  - $p[v] = -1$
- marque todos os vértices como "abertos" (o valor de d é um chute/estimativa)

- enquanto houver vértice aberto:
  - escolha o vértice aberto u cuja a estimativa seja a menor dentre os demais abertos.
  - feche u
  - para todo nó aberto v na adjacência de u:
    - some $d[u]$ ao peso da aresta (u,v)
    - caso a soma seja menor que $d[v]$, atualiza $d[v]$ e faça $p[v] = u$

procedimento chamado de relaxamento da aresta (u,v)

observação: os pesos das arestas devem ser não-negativos, pois o algoritmo pressupõe que uma vez que o vértice é fechado, não há um caminho menor para chegar até ele, assim, se há uma aresta com peso negativo na frente que faça o caminho de um certo vértice anterior ser menor, esse caminho não será atualizado, pois o vértice já estará fechado.

COMPLEXIDADES:

dfs → O(V+E)

bfs → O(V+E)

kmp → O(n)

tempo para construir a matriz → O(m * |alfa|) → não sei o que é isso

MAC0323 - ALGORITMOS E ESTRUTURAS DE DADOS II

ALGORITMOS DE PROCESSAMENTO DE TEXTOS

ALGORITMO KMP

- examina os caracteres de txt um a um, da esquerda para a direita, sem nunca retroceder.
- em cada iteração, o algoritmo sabe qual posição k de pat deve ser emparelhada com a próxima posição $i+1$ de txt.
- o algoritmo KMP usa uma tabela dfa[][] que armazena os índices mágicos k
- o algoritmo KMP simula o funcionamento do autômato de estados.
- faz $O(n)$ comparações (n é o tamanho do texto)
- roda em $O(n)$ mais o tempo para construir a matriz dfa
- tempo para construir a matriz: mR (m = tam do padrão, R = tam do alfabeto)

```
int KMP(char * texto, char * padrao){
   int m = strlen(padrao);
   int n = strlen(texto);
   int i ,j; /* i percorre o texto, j percorre o padrao*/
   for(i=0,j=0; i<n && j<m; i++)
      j = dfa[texto[i]][j];
   if(j==m) return i-m;
   return; /*nao ocorre*/
}
```

EXPRESSÕES REGULARES

- buscar por palavras que comecem de um jeito, terminem ou contenham algum padrão.
- **CONCATENAÇÃO:** se x_1 e x_2 são exp. reg., $x_1 x_2$ é uma exp reg.
- **ALTERNATIVA / OU:** $x_1 | x_2$
- **FECHO:** x_1^*, reconhece 0 ou mais cópias de p_1
  - vazio = ϵ
  - $\cup$
  - . : coringa

    Ex: AB.C = {ABAC, ABBC, ...}

    AB(A|B|C|...|z)C
  - \+ : fecho com pelo menos uma cópia
  - ? : zero ou uma cópia

    B(A?)B = BB, BAB
  - {k} : número fixo de repetições

    A{3}CB{2}DA{5} = AAACBBDAAAAA
  - [ ] : conjunto

    R[AEIOU]S[AEIOU]*
  - [-] : intervalo

    [0-9]*\-[0-9]* : 4123-15, 32-7, -4, 4-
  - [^] : complemento

    [^0-9][0-9]* : A234, B, A11...

Expressões regulares podem ser usadas como poder de expressão:

EXEMPLO: ([a-z]|[0-9]|\.|_)+@(([a-z])+\.)+br

aaa.2345.f2@ime.usp.com.br

\([0-9]{2}\) 9[0-9]{4}\-[0-9]{4}

(11) 97143-1234