- Teórica 9:

Os Grafos são estruturas não lineares de dados – nas relações entre os dados não existem restrições para ir de um vértice para qualquer outro vértice.

Um Grafo possui um conjunto de vértices (V) e ligações (L) em que cada ligação é um conjunto de dois vértices distintos. Um caminho é uma sequência de vértices diferentes em que cada par sequencial de vértices possui uma ligação.

Um ciclo é um caminho em que o primeiro e o último vértice são o mesmo. Um Grafo diz-se conexo se existir um caminho de qualquer vértice para qualquer outro vértice do grafo.

- Grafo Orientado: um Grafo Orientado é um conjunto de vértices (V) e ligações (L) em que cada ligação é uma sequência de dois vértices (Ex: <3,6>). Um caminho é uma sequência de vértices em que cada par sequencial de vértices possui uma ligação (Ex: <1,3,6,4>).
- Grafo Ponderado: Grafo a cujas ligações podem ser atribuídas pesos. Se o Grafo é orientado e ponderado toma o nome de Rede.

- Grafos – Propriedades:

Um Grafo com um número V de vértices tem, no máximo, V(V-1)/2 ligações.

Um Grafo conexo sem ciclos é uma Árvore – Um conjunto de Árvores chama-se Floresta.

Um Árvore Abrangente (Spanning Tree) de um Grafo Conexo é um subgrafo que contém todos os vértices do Grafo e é uma Árvore;

- Grafos – Metodologias de Representação:

1. Matriz de Adjacência: permite fácil acesso (e inserção e/ou eliminação) às ligações, ou seja, permite uma resposta fácil à pergunta: há ligação entre "i" e "j"? Utiliza memória em excesso em Grafos com muitos vértices e poucas ligações;
2. Lista de Adjacência: o acesso (e inserção e/ou eliminação) às ligações não é tão fácil. Contudo, o desenho de algoritmo que não necessitem deste acesso é fácil, sendo normalmente utilizado um método de procura para visitar todos os vértices (e respetivas ligações) e proceder, eventualmente, às alterações necessárias;

As operações típicas em Grafos são: Inserção de Vértices e/ou Ligações, Eliminação de Vértices e/ou Ligações, Pesquisar/Percorrer o Grafo, Conetividade, Caminho mais curto/longo, Minimum Spanning Trees e Fluxo de Redes;

Algumas das aplicações dos Grafos são sistemas de navegação, hipertexto e motores de busca, circuitos, redes e modelação/estudo de sistemas (dinâmicos);

- Grafos – Pesquisa:

O termo pesquisa é geralmente empregue, no âmbito dos Grafos, no processo que examina (explora) todos os seus vértices. Quase todos os algoritmos de Grafos utilizam este processo de saltar de vértice em vértice através das ligações existentes.

Existem 2 processos típicos para pesquisar um Grafo conexo:

1. Pesquisa baseada na Profundidade (Depth-First-Search) – Tenta sempre ir mais fundo: Explora todas as ligações dos vértices da Árvore construída a partir das ligações de um Vértice V. Se permanecem Vértices por visitar, recomeça o processo utilizando um desses vértices;
   1.1. Possui duas implementações: Uma recursiva e uma através da utilização de uma Stack;
   1.2. Pior Caso (Com Listas de Adjacência): O(|E| + |V|) – Uma vez que todos os vértices e ligações serão utilizados no pior dos cenários;
   1.3. Pior Caso (Com Matrizes de Adjacência): O(|V|^2) – Uma vez que se torna necessário percorrer toda a Matriz;
   1.4. Classificação de Ligações: As ligações que representam uma chamada recursiva (Tree Edges) e as ligações entre um vértice e um antecessor na Árvore DFS que não seja o seu pai (Back Edges);
   1.5. Um Back Edge de uma árvore DFS pertence a um ciclo composto por essa ligação mais o caminho da Árvore que liga os dois vértices;
   1.6. Um Grafo não tem ciclos se não existirem Back Edges (Back Links ou Down Links) na Árvore DFS;
2. Pesquisa baseada na Amplitude (Breadth-First-Search) – Explora uniformemente a partir de um vértice de partida: Visita todos os vértices adjacentes ao vértice de partida, estabelecendo deste modo um conjunto de vértices. Utiliza cada elemento deste conjunto para ponto de partida;
   2.1. Pior Caso (Listas de Adjacência): O(|E| + |V|) – Uma vez que todos os vértices e ligações serão utilizados no pior dos cenários;
   2.2. Pior Caso (Matrizes de Adjacência): O(|V|) e O(|V^2|), dependendo do número de ligações do Grafo e do tipo de representação;

- Grafos: Caminho Mais Curto

A Árvore de Pesquisa, resultante do método de pesquisa baseada na amplitude (BFS), pode ser utilizada na determinação de caminhos mais curtos.

1. Caminho mais curto de um vértice V para um vértice W: O procedimento de pesquisa é interrompido quando se visita W;
2. Caminhos mais curtos a partir de um vértice V: A Árvore resultante do procedimento de pesquisa contém os caminhos mais curtos;
3. Caminhos mais curtos de todos os pares de vértices: Repete-se o método para todos os vértices de Grafo;

- Grafos Ponderados:

Uma Árvore Abrangente de Custo Mínimo (Minimum Spanning Tree) de um Grafo Ponderado é uma Árvore Abrangente cujo custo (soma dos custos das suas ligações), é menor ou igual ao custo de qualquer outra Árvore Abrangente desse Grafo.

Os custos das ligações podem ser negativos. Se houver custos iguais em ligações diferentes, a MST pode não ser única.

- Algoritmo de Prim:

O método base baseia-se na ideia de manter um corte do Grafo, dividindo-o em vértices da MST e vértices não pertencentes à MST. Adequado para Grafos densos.

Numa versão melhorada, pode ser considerado uma "extensão" do algoritmo BFS (utiliza uma fila de espera com prioridades em vez de uma fila de espera).

Eficácia (O Grafo está representado com uma Matriz de Adjacência):

1. Com pesquisa baseada num vetor de pesos de ligações: O(V^2), eficaz para Grafos densos;
2. Com pesquisa baseada em filas de espera com prioridades (Heap): O(E log V), limite superior satisfatório;
3. Com pesquisa baseada em filas de espera com prioridades (D-Heap): O (E + V log V), custo linear a não ser que a matriz seja extremamente esparsa;

- Algoritmo de Kruskal:

O método de Prim constrói a MST adicionando uma ligação de cada vez (a ligação ponte entre os conjuntos resultantes do corte do Grafo).

Por sua vez, o método de Kruskal também constrói a árvore ligação a ligação, mas fá-lo encontrando a ligação que une duas Árvores numa floresta de MSTs.

O algoritmo inicia-se com uma floresta de V Árvores (cada uma contendo unicamente um vértice) e com a ordenação das ligações por custo.

Eficácia (O Grafo está representado utilizando uma matriz de adjacência):

1. Com pesquisa baseada numa estrutura simples de dados: O(E log E);
2. Com a utilização de filas de espera com prioridades: O(E + XlogV), em que X é o número de ligações cujo custo é menor do que o maior custo de uma ligação da MST;

- Grafos Orientados e Ponderados:

O caminho mais curto entre dois vértices S e T para o caso de Redes, é o caminho orientado de S para T cujo custo não seja maior do que o custo de qualquer outro caminho de S para T;

Para o caso de Redes o Algoritmo de Prim (utilizando filas de espera com prioridades) para MSTs pode ser modificado, no sentido de comportar os pesos das ligações. O problema passa a ser encontrar o caminho menos pesado entre dois vértices.

Esta solução para o problema é designada por Algoritmo de Dijkstra. O algoritmo utiliza uma estratégia de menor custo (Greedy) na visita dos diferentes vértices ao longo do processo.

- Algoritmo de Dijkstra:

O Algoritmo de Dijkstra funciona visitando vértices no Grafo a partir de um ponto de partida. Examina repetidamente o vértice mais próximo (da origem) ainda não examinado, adicionando os seus vértices ao conjunto de vértices a serem examinados (expande-se a partir do ponto de partida até que atinja o objetivo).

O Algoritmo de Dijkstra garante que encontra um caminho mais curto de um ponto de partida para um outro vértice, desde que nenhuma das ligações tenha custo negativo.

Eficácia do Algoritmo de Dijkstra:

1. Standard – V^2: Eficaz para Grafos densos;
2. Implementado com Full-Heap – E log V: Implementação mais simples;
3. Implementado com Fringe Heap – E log V: Limite superior satisfatório;
4. Implementado com D-Heap – E log(d) V: Linear, a não ser que a matriz seja extremamente esparsa;

- Grafos: Caminho Mais Curto

A pesquisa do caminho mais curto é complexa – principalmente quando envolve obstáculos. Se se avaliar uma área mais alargada (“pathfinding” em vez de “movement”) normalmente deteta-se um caminho mais curto nestas situações (planeamento prévio invés de movimento). Deve-se fazer a escolha em função do problema.

O Algoritmo de Dijkstra é correto (optimal solution) mas demasiado lento para algumas situações – a razão tem a ver com o número de vértices da designada Fringe (conjunto dos vértices ainda não visitados mas que o poderão ser no próximo passo, por serem adjacentes aos vértices já visitados) que o método analisa/visita.

A solução passa pelo uso de Heurísticas (método ou processo criado com o objetivo de encontrar soluções para um problema. É um procedimento simplificado que, em fase

de questões difíceis, envolve a substituição destas por outras de resolução mais fácil a fim de encontrar respostas viáveis, ainda que imperfeitas) para diminuir esse número de vértices.

A utilização de Heurísticas em algoritmos de caminho mais curto permite diminuir o número de vértices analisados em cada iteração e, desse modo, tornar o processo mais rápido. Uma Heurística diz-se admissível se, para cada vértice n – h(n) <= h*(n) em que h*(n) é o custo correto para atingir o destino pretendido a partir de n.

Para tal, utiliza-se uma função f(n) em cada vértice que dá uma estimativa do custo total. Perante diferentes valores de f(n) nos diferentes vértices a visitar, escolhe-se o vértice de menor valor de f(n) (Algoritmo Greedy).

A implementação passa por ordenar os vértices da Fringe por ordem decrescente deste valor. Existem dois algoritmos que utilizam esta abordagem: Greedy Best-First Search e A* Search;

No caso do Algoritmo Greedy Best-First Search, f(n) = h(n), em que h(n) pode ser a distância em linha reta de n até ao objetivo. Deste modo, escolhe-se o vértice que parece estar mais perto do objetivo.

No caso do Algoritmo A* Search, f(n) = g(n) + h(n), em que g(n) é o custo real da origem até n e h(n) pode ser a distância em linha reta de n até ao objetivo. Deste modo, não se escolhe caminhos que já provaram não ser os melhores. Se h(n) é admissível, o Algoritmo A* (se utilizar uma pesquisa baseada em árvores binárias) apresenta uma solução ótima.

- Algoritmo de Dijkstra: o caminho obtido é o melhor, mas é algo lento para grafos com uma Fringe muito extensa;
- Algoritmo Best-First Search: o caminho obtido não é o melhor. O problema resido no fato de não o contabilizar o caminho já percorrido (não eliminando caminhos muito longos) e avaliar o custo pela distância ao objetivo;
- Algoritmo A* Search: o caminho obtido é o melhor e é rápido a calcular;

A abordagem BFS designa-se por pesquisa informada, que expande o nó mais promissor em primeiro lugar. O Algoritmo A* é um caso particular desta abordagem.

A maior desvantagem desta abordagem (e logo do Algoritmo A*) é a grande necessidade de recursos de memória.

O Algoritmo de Dijkstra é designada um algoritmo de pesquisa não informada (pesquisa exaustiva), que deve ser utilizado quando não possuímos informação prévia sobre o Grafo.

O Algoritmo A* resulta no Algoritmo de Dijkstra quando a função heurística h(n) = 0 para cada nó existente.

- Algoritmo BFS: não é completo, pois nem sempre encontra uma solução, podendo ficar preso em ciclos (dependendo da função f(n)). Gasta O(b^m) de

tempo (uma boa Heurística pode operar melhorias significativas) e O(b^m) de memória, pois mantém todos os nós guardados, em que b é o número de ramificações e m o nível de ramificações existentes;

- Algoritmo A*: é completo, a menos que haja um número infinito de nós em que f <= f. Gasta O(b^m) exponencial de tempo e O(b^m) de memória. É um algoritmo ótimo com uma eficácia ótima.

- Teórica 11:

A inteligência de enxame é um tipo de computação recente, baseada numa metáfora, para resolver problemas distribuídos. É capaz de desempenhar tarefas difíceis em ambientes sem qualquer auxílio, controlo externo ou coordenação central.

- Vantagens: Feedback positivo na descoberta rápida de boas soluções. Computação distribuída evita convergência prematura. A heurística gulosa ajuda a encontrar uma solução aceitável na solução rápida nas fases inicias do processo de pesquisa. Interação coletiva de uma população de agentes.
- Comportamento comuns: O controle é totalmente distribuído entre um número de indivíduos. A comunicação entre os indivíduos acontece numa forma localizada. O comportamento do sistema apresenta transcender o comportamento individual. A resposta do sistema é bastante robusta e adaptativa em relação às mudanças do sistema.

- Sistema ACO:

Os rastos virtuais de feromona são acumulados nos caminhos. O nodo inicial é escolhido aleatoriamente. O caminho é selecionada aleatoriamente, baseado na quantidade de feromona presente no caminho possível desde o nodo inicial e existe uma maior probabilidade para caminhos com mais feromona. A formiga ao chegar ao nodo seleciona o próximo arco a seguir. Continua até alcançar o nodo destino: uma volta completa é uma solução possível.

Os rastos de feromona são ajustados de modo a favorecer as melhores soluções – as melhores recebem mais feromona, as piores recebem menos e existe maior probabilidade das formigas escolheres caminhos que pertençam às melhores voltas. Um novo ciclo é executado e o processo é repetido até a maioria das formigas escolher o mesmo caminho em cada novo ciclo (convergência do algoritmo).

O algoritmo ACO é usado normalmente para resolver problemas de minimização (N nodos e A arcos).

Passos para implementar um algoritmo ACO:

1. Representar o problema em forma de conjuntos de componentes e transições, ou através de um grafo ponderado, no qual as formigas podem construir soluções;

2. Definir o significado dos rastos de feromona;
3. Definir uma heurística para as formigas enquanto constroem a solução;
4. Se possível, implementar um método de pesquisa local;
5. Escolher um algoritmo ACO;
6. Especificar os valores dos parâmetros do algoritmo;

No fim de cada iteração, guarda-se o melhor percurso e a população é substituída.