Documento: **Resumo do Conteúdo da P1**
Disciplina: **ACH2024 - Algoritmos e Estruturas de Dados II**
Prof: **Helton Hideraldo Biscaro**

Histórico:
versão 1, 01/10/2012, Henrique Leme

Grafos

Existem centenas de problemas recorrentes que podem ser resolvidos com grafos. Vamos ver exemplos de problemas e diversos tipos de algoritmos.

Definições básicas:

Grafo (graph): modelo matemático para representar relacionamento entre objetos.

$$G=(V,E)$$

Vértices (vertice): são os objetos do relacionamento.

$$V=\{v_1, v_2, ... v_n\}$$

Arestas (edge): são o relacionamento entre os objetos (vértices).

$$E=\{e_1, e_2, ... e_m\} \quad \text{onde} \quad e=(v_1, v_2)$$

$$E=\{(v_1, v_2), (v_1, v_5)...\} \quad \text{ou} \quad E=\{\{v_1, v_2\}, \{v_1, v_5\}...\}$$

Representação Visual: os vértices são pontos e as arestas são linhas, setas ou arcos.

Aplicações:

Muito utilizado para modelar sistemas reais como por exemplo:

- redes de distribuição de energia ou telecomunicação
- malhas viárias
- circuitos elétricos
- processos industriais e processos de negócio
- mapas de rotas aéreas, rodoviárias, etc.
- coloração de mapas
- teoria dos jogos

Definições complementares:

Ordem: é a cardinalidade do conjunto de vértices $|V|$ do grafo G

$$G=(V,E)$$
$$V=\{v_1, v_2, ... v_n\}$$
$$|V|=n$$

Número de arestas: é a cardinalidade do conjunto de arestas $|E|$ do grafo G

$$G=(V,E)$$
$$E=\{e_1, e_2, ... e_m\}$$
$$|E|=m$$

Laço: Uma aresta com que inicia e termina no mesmo vértice.

$$e=(u,v) \quad \text{tal que} \quad u=v$$

Arestas paralelas: Duas arestas que inicial e termina nos mesmos vértices.

$$e_1=(u_1, v_1) \quad \text{tal} \quad e_2=(u_2, v_2) \quad \text{tal que} \quad u_1=u_2 \quad \text{que} \quad v_1=v_2$$

Grafo simples: grafo sem laços ou arestas paralelas.
Multigrafo: grafo com arestas paralelas porém sem laços.
Pseudografo: grafo com arestas paralelas e com laços.
Vértices adjacentes: são dois vértices conectados por uma aresta.
Arestas adjacentes: são duas arestas que possuem um vértice em comum.
Arestas incidente: a aresta é incidente a cada um dos vértices conectado por ela.
Grafo nulo: quando o número de arestas é igual a zero.
Grafo completo: existe exatamente uma aresta para cada par de vértices distintos.

seja $|V|=n$ o número de arestas é $x=\frac{(n^2-n)}{2}$

Grafo valorado: as areas contém um valor (peso) associado.

$e=(u,v,x)$ onde x é o valor.

Grafo direcionado: as areas passa a ser um par ordenado de vértices.

$e=(v_{1,}v_2)$ sendo $(v_{1,}v_2)$ diferente de $(v_{2,}v_1)$

Aresta convergente sobre um vértice: aresta que sai deste vértice.

Aresta divergente sobre um vértice: aresta que chega neste vértice.

Grau de um vértice: número de arestas incidentes.

$d(v_1)$

Grau de um grafo: soma dos graus de todos os vértices.

$\sum d(v)=2|E|$

Definições de caminhos e circuitos:

Vértice isolado: vértice de grau zero.

Vértice pendente: vértice de grau 1.

Vértice ímpar: vértice de grau ímpar.

Vértice par: vértice de grau par.

Grafo regular (k-regular): grafo onde todos os vértices tem o mesmo grau (k).

$\sum d(v)=k|V|$

Grau de saída: é o número de arestas divergente de um vértice

$d^{+.}(v)$

Grau de entrada: é o número de arestas convergentes sobre um vértice.

$d^{-.}(v)$

Fonte: é um vértice que só tem arestas divergentes.

$d^{-.}(v)=0$

Sumidouro: é um vértice que só tem arestas convergentes.

$d^{+.}(v)=0$

Grafo bipartido: quando os vértices estão divididos em dois grupos e cada áreas tem um vértice de um grupo e o outro do outro grupo.

Grafo Npartido: cada um dos N subgrafos não tem arestas onde os dois vértices estão dentro do próprio subgrafo.

Subgrafo: um grafo $G'=(V',E')$ é subgrafo de $G=(V,E)$ quando $V'\subset V$ e $E'\subset E$.

Grafo complementar: dois grafos quando eles em os mesmos vértices e o conjunto das suas arestas somadas fazem um grafo completo.

Caminho: é uma sequência de vértices e áreas alternados tal que cada aresta é incidente ao nó anterior e ao nó posterior.

$(v=)v_0e_1v_1...v_n-1\,e_nv_n(=w)$

Extremidade do caminho: são as arestas inicias e finais de um caminho.

Caminho fechado: é um caminho que inicia e termina no mesmo vertice.

$v_0=v_n$

Caminho trivial: é um caminho do vértice v para o vértice v (sem arestas).

Ciclo: é um caminho fechado que tem pelo menos uma aresta.

Grafo acíclico: Grafo que não tem ciclos.

Comprimento do caminho: Número de arestas percorridas pelo caminho.

Caminho simples: Caminho que não tem vértices repetidos.

Circuito: é um caminho fechado sem arestas repetidas.

Circuito simples: é um ciclo (caminho fechado) sem arestas repetidas.

Grafo conexo: todos os vértices são alcançáveis por um caminho simples a partir de qualquer outro vértice.

Componente Conexo: é o subgrafo conexo de maior tamanho.

Grafo fortemente conexo: é um grafo direcionado onde todos os vértices são alcançáveis por um caminho simples a partir de qualquer outr vértice seguindo a orientação das arestas.

Grafo Euleriano e Hamiltoniano:

Caminho Euleriano: é o caminho que percorre cada areas apenas 1 vez.

Circuito Euleriano: ou grafo Euleriano, é o circuito que passa por todas as arestas exatamente 1 vez. Teorema:

- G tem que ser conexo
- Todo vértice tem que ter grau par.

Circuito Hamiltoniano: ou grafo Hamiltoniano, é o circuito que passa por todas os vértices exatamente 1 vez (exceto o final/inicial). Não existe forma polinomial de encontrar um circuito hamiltoniano.

Se G tem um circuito Hamiltonano se G tem um subgraf H com a seguite propriedades:

- H contém todos os vértices de G
- H é conexo.
- H tem mesmo número de arestas e vértices.
- Cada vértice de H tem grau 2.

Problemas e soluções:

Problema do caixeiro viajante: pode ser representado por um grafo valorado, direcionado ou não. O problema é resolvido por um circuito Hamiltoniano com o menor custo.

Pontes de Konigsberg: pode ser representado por um grafo não valorado e não direcionado. O problema teria que resolvido por um circuito Euleriano, porém prova-se que não tem resposta.

Jogo de Hamilton: pode ser representado por um grafo não valorado e não direcionado. O problema pode ser resolvido por um circuito Hamiltoniano.

Problema do carteiro chinês: pode ser representado por um grafo valorado e não direcionado. Duplica-se as arestas de menor custo até que o grafo se transforme em um circuito Euleriano. O problema é resolvido por um circuito Euleriano com o menor custo.

Ordenação Topológica: A ordenação topológica só funcionam para gráficos DAG (gráficos direcionados e acíclicos). É uma ordenação linear de todos os vértices tal que se o grafo G tem uma aresta orientada (u, v) então, na lista, u aparece antes de v.

Coloração de grafos: Todos os vértices adjacentes tem que ter cores diferentes. A quantidade de cores do grafo é a quantidade de partições necessárias (grafo Npartido). Todo grafo planar admite 4-coloração, ou seja todo grafo planar é quadri-partido.

Isomorfismos de Grafos:

Os grafos são representados por uma lista de vértices e lista de arestas. Um mesmo grafo pode ter diversas representações gráficas (na forma de um diagrama).

Propriedade reflexiva: um grafo é isomorfo a si próprio.
Propriedade simétrica: se G é isomorfo a G' então G' é isomorfo a G.
Propriedade transitiva: se G é isomorfo a G' e G' é isomorfo a G" então G é isomorfo a G".

Algoritmos:

Busca em largura: Percorre todas as arestas do grafo visitando todas as arestas de cada vértice encontrado antes de visitar o vértice oposto de cada aresta. Tem que ter um vetor de vértices visitados para não entrar em loop.

Busca em profundidade: Percorre todas as arestas do grafo visitando o vértice oposto de cada aresta antes de visitar as arestas adjacentes.Tem que ter um vetor de vértices visitados para não entrar em loop.

Vertice de corte: é um vertice, que ao ser removido aumenta o número de componentes conectados. Algoritmo básico com complexidade $O(VE)$:

• calcula o número dos componentes conectados.
• para cada vértice $u \in V[G]$
• remove o vértice u
• re-calcula o número dos componentes conectados.

Algoritmo melhorado:

• A busca em largura (BFS) pode ser modificada para complexidade $O(V+E)$

Aresta ponte: é uma aresta, que ao ser removida aumenta o número de componentes conectados.

K-vértice-conexo: é um grafo que permanece conexo quando menor que k vértices são removidos do grafo.

Corte máximo: Sobre um grafo valorado, procurar um conjunto de vértices $S \in V$ e $T = S^c$ que fazem com que a soma dos valores das arestas incidentes em S e T seja máximo.

Memória secundária

A memória secundária é muito lenta. Precisamos entender um pouco da organização da memória secundária para entender seus gargalos e adotar estratégias para otimizar o seu acesso. O ideal é fazer um único acesso para leitura e/ou um único acesso para gravação. Nem sempre isso é possível, portanto tempos alguns algoritmos úteis para deixar estes acessos o mais próximo do ideal.

Estruturas de dados:

Árvore Simples: pode ficar desbalanceada e virar uma lista.
Árvore AVL: acesso razoável O(log n) porém requer um trabalho para se manter balanceada.
Árvore B: melhor que a AVL porém não fornece ordenação dos dados.
Árvore B+: tem uma lisa encadeada no nível inferior da árvore B.
Hashing: recuperação de dados com 1 acesso porém não fornece ordenação.
Hashing dinâmico: pode ser aplicado a arquivos dinâmicos eficientemente.

Dispositivos físicos de armazenamento:

Setor: menor porção endereçável de um disco. Historicamente tem 512 bytes.
Trilha: conjunto de setores que estão acessíveis, em uma única cabeça de leitura sem que haja movimento físico.
Cilindor: conjunto de trilhas (todas as cabeças de leitura) que estão acessíveis sem que haja movimento físico da cabeça de leitura.
Disco: conjunto de cilindros.
Latência de busca (seek): tempo médio de espera para que a cabeça chegue na trilha (seg).
Latência de rotação: tempo médio de espera para que o início setor chegue até a cabeça (seg).
Taxa de transferência: tempo médio de transferência de dados em bytes por segundo.

Dispositivos lógico de armazenamento:

Setor: menor porção lógica endereçável, geralmente do mesmo tamanho que o setor físico.
Cluster (blocos): conjunto de setores lógicos adjacentes.
Arquivo: conjunto de clusters interligados por uma lista.
Extends: é um grupo de clusters consecutivos.
Fator de bloquamento: quantidade de setores por bloco.

Tempo de acesso (leitura ou gravação):

Seja S quantidade de dados transferidos em bytes; E quantidade de extends; T_t a taxa de transferência em byte / segundo; L_s a latência de busca (seek) em segundos; L_r a latência rotacional em segundos; o resultado é T tempo de acesso em segundos;

$$T = \frac{S}{T_t} + E\left(L_s + L_r\right)$$

Ordenação de Listas Grandes

A ordenação diretamente na memória secundária seria possível, porém, dado que ela é muito lenta, é bem dispendiosa. Podemos passar os dados para a memória principal, fazer a ordenação e depois devolver para a memória secundária. Porém, se a lista for realmente grande, ela não vai caber totalmente na memória principal. A seguir tem algumas técnicas para ordenar listas grandes.

Keysort:

Não carregar todo o registro para a memória. Ordenar apenas o índice e depois movimentar os registros uma única vez.

Merging (k-vias ou k-ways):

O arquivo é quebrado e pedaços que caibam na memória. Cada pedaço é ordenado na . Todos os pedaços são intercalados para recompor o arquivo ordenado.

Fase de ordenação: o arquivo é dividido em pedaços que caibam na memória, cada pedaço é ordenado e gravado de volta para disco. Cada execução da fase de ordenação, e o respectivo arquivo em disco, é chama de run (corrida). A ordenação tem a etapa de leitura e a etapa de gravação.
Fase de intercalação: cada pedaço é intercalado para gerar o arquivo final ordenado. A intercalação tem a etapa de leitura e a etapa de gravação.
Tabela de custo de Merging: precisamos calcular o custo de cada etapa do processo: leitura da ordenação, gravação da ordenação, leitura da intercalação e gravação da intercalação. Cada etapa tem os tempos de latência de busca (seek), latência de rotação e taxa de transferência.
Multistep Merging: Podemos executar este merging em vários níveis para deixar o processo mais rápido. A fase de leitura de merging é $O(n^2)$ e as outras etapas são $O(n)$ portanto pode haver um balanceamento com o resto das outras fases para melhorar a performance.

Replacement Selection:

Outra forma de quebrar o arquivo em pedaços que caibam na memória, porém com menos arquivos intermediários. Faz uso de duas lista ordenadas em memória.

Fase de ordenação:

a) o arquivo de entrada é lido para a primeira lista ordenada até que a memória fique cheia.
b) o menor registro da primeira lista em memória é gravado em um arquivo de corrida (run) em disco.
c) para cada registro gravado mo arquivo de corrida (run), um novo registro é lido da entrada, se for maior que o último registro gravado é colocado na primeira lista se for menor é colocado em uma segunda lista ordenada.
d) quando a primeira lista ordenada estiver vazia, a lista secundária passa a ser a primária, a lista primária (vazia) passa a ser a secundária e a gravação de saída passa a ser feita em um segundo arquivo de corrida (run).
e) estes passos são seguidos até que a lista de entrada termine.

Fase de intercalação:

f) depois é só fazer o merge dos arquivos de corrida (runs).

Hash

Uma forma mais eficiente de recuperar dados (sem ordenar) é usar algoritmos de hashing. O hashing é uma função que retorna um índice de um vetor a partir de uma chave. Uma vez calculado o hash de uma chave, o acesso, hipotético, é com complexidade $O(1)$.

Definições:

Vetor de chaves: é um vetor que caibam as chaves de todos os registos.
Tamanho do vetor: T_s é o tamanho do vetor chaves (table size).
Chave: c chave do registro, sendo que $c \in C$ pertence ao conjunto de chaves.

Índice: i índice no vetor de chaves, sendo que $i \in I$ pertence ao conjunto de índices.
Função hash: são os objetos do relacionamento.

$$h : C \rightarrow I$$
$$h(c) = i$$

Vantagens e Desvantagens:

Vantagens: simplicidade e velocidade.
Desvantagem: a) colisão pois o hash (por definição) não é uma função injetora; b) não ordena; c) desperdiça espaço; d) exige conhecimento da chave e da distribuição.

Problema de colisão:

Colisão: como hash, por definição, não é uma função injetora, duas chaves distintas tem alguma chance de ter o mesmo índice hash.
Rehash: geramos uma função de rehash para achar um outro índice para esta chave com colisão. Por exemplo, o rehash mais simples, o linear $rh(i) = (k + e)\%T$.
Problemas do rehash: o vetor de dados não pode encher (tem que ter pelo menos 1 item vazio). A função de remoção de chave tem que gravar uma marca de item apagado.

Problema de rehash:

Vetor cheio: o vetor de chaves não pode encher (tem que ter pelo menos 1 item vazio). Se ele encher, a rotina de busca para uma chave
Remover chave: A função de remoção de chave tem que gravar uma marca de chave apagada diferente da marca de chave inexistente. Quando uma chave é apagada no meio da lista de colisão, recebe uma marca de apagado para que as chaves seguintes na lista de colisão possam ser encontradas.

Aglomeração primária:

Probabilidade do índice: inicialmente, a probabilidade de um índice ser selecionado é igual para todas as posições $P(i) = \frac{1}{T}$.
Aglomeração primária: depois que uma chave é colocada, a probalidade de alguns índices passam a ser maiores quando eles são a próxima chave vazia em uma lista de colisões. O rehash linear gera uma aglomeração primária bem grande.

Eficiência do hash:

Load factor: $L_f = \frac{m}{T_s}$

Probe success: $P_s = \frac{(2 - L_f)}{(2(1 - L_f))}$

Probe fail: $P_f = (\frac{1}{(1 - L_f)^2}) + (\frac{1}{2})$

Solução para aglomeração primária:

Tabela de permutação: fazer uma tabela de permutação que faz o papel do rehash.
Rehash com dois parâmetros: por exemplo: $rh(c, j) = (h(c) + \sqrt{j})\%T$. Porém gera uma aglomeração secundária.
Novo rehash: várias rotinas diferentes para rehash().
Split: duas rotinas lineares de rehash com duas constantes distintas. Só na primeira colisão, $h(k) < k$ usa a primeira constante, senão usa a segunda constante. O problema é remover uma chave que é a primeira na lista de colisões, não sabemos de devemos usar a constante 1 ou 2.