UENF

Universidade Estadual do Norte Fluminense Darcy Ribeiro

Curso: Ciência de Computação **Data:** 08./.09./2022
Trabalho: Trabalho1 **Período:** 7º **Disciplina:** Top. Esp.: Heurísticas e Complexidade
Professor: Fermín Alfredo Tang **Turno:** Diurno
Nome do Aluno: **Daniel Terra Gomes** **Matrícula: .....**

1.- Com base na Tabela 1, que apresenta alguns problemas da classe NP-Completo, escolha 5 problemas. Obs. Outros problemas além da tabela podem ser escolhidos caso sejam de interesse.

Tabela 1.- Problemas NP-Completos

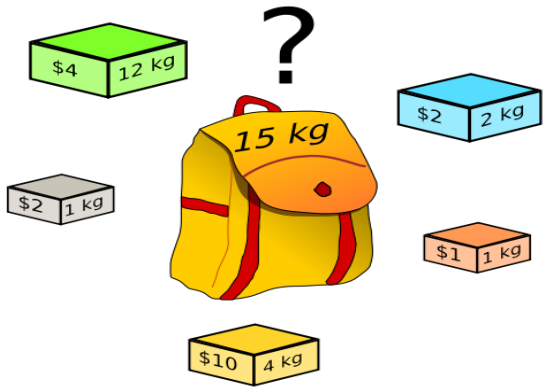
#	**Problemas NP-Completos**	**NP-Complete Problem**
1	SAT	*Satisfiability Problem*
2	3SAT	*3-Satisfiability Problem*
3	Caminho mais Longo/ ou Crítico	*Longest Path Problem/ Critical Path*
4	Mochila	*Knapsack Problem*
5	Empacotamento	*Bin Packing Problem*
6	Casamento 3D	*3-Dimensional Matching Problem*
7	Conjunto Independente	*Independent Set Problem*
8	Cobertura por Vértices	*Vertex Cover Problem*
9	Clique	*Clique Problem*
10	EZU – Equações Zero Um	
11	Cobertura de Conjuntos	*Set Covering Problem*
12	Soma de Conjuntos	*Subset Sum Problem*
13	Programação Linear Inteira	*Linear Programming Problem*
14	Caminho Rudatra	*Rudatra Path*
15	Ciclo Hamiltoniano / Caixeiro Viajante	*Hamiltonian Cicle/ Traveling Salesman Problem*
16	Corte Balanceado	*Balanced Cut Problem*
17	Graph coloring/Vertex Coloring	*Graph coloring/Vertex Coloring*

Para cada um dos problemas escolhidos, responda às seguintes questões:

i) Defina o problema mediante uma descrição verbal, em palavras. Pesquise sobre uma definição formal usando notação matemática e/ou de grafos para o problema escolhido;

Problema da mochila (alocação de recursos otimizada)

O problema da mochila refere-se ao problema comum de embalar seus itens mais valiosos ou úteis sem sobrecarregar sua bagagem. Ele deriva seu nome do problema enfrentado por alguém que é limitado por uma mochila de tamanho fixo (tamanho da mochila) e deve enchê-la com os itens mais valiosos (Koroth).



1. Ilustração Problema da Mochila (“Knapsack problem”).

Sendo um exemplo de problema de otimização combinacional, um tópico em matemática e ciência da computação sobre encontrar o objeto ideal entre um conjunto de objetos. Este é um problema que vem sendo estudado há mais de um século e é um problema de exemplo comumente usado em otimização combinatória, onde há a necessidade de um objeto ótimo ou solução finita onde uma busca exaustiva não é possível. O problema pode ser encontrado em cenários do mundo real como alocação de recursos em restrições financeiras ou até mesmo na seleção de investimentos e portfólios. Também pode ser encontrado em áreas como matemática aplicada, teoria da complexidade, criptografia, combinatória e ciência da computação. É facilmente o problema mais importante na logística (“What is the Knapsack Problem? - Definition from Techopedia”).

Formalmente, suponha que recebemos os seguintes parâmetros:

wk = o peso de cada item do tipo k, para k = 1, 2, . . ., N,
rk = o valor associado a cada item do tipo k, para k = 1, 2, . . ., N,
capacidade de peso da mochila.

Então, nosso problema pode ser formulado como:

$$\sum_{k=1}^{N} r_k x_k$$

Maximizar

Sujeito a: $\sum_{k=1}^{N} w_k x_k \quad \leq \quad c\,,$

onde x1, x2, . . ., xN são variáveis de decisão de valor inteiro não negativo, definidas por

xk = o número de itens do tipo k que são carregados na mochila.

Observe que, como os xk são de valor inteiro, o que temos não é um programa linear comum, mas sim um programa inteiro (“The Knapsack Problem”).

Caminho mais Longo/ ou Crítico

O problema do caminho mais longo pede para encontrar um caminho de comprimento máximo em um determinado grafo. O problema é NP-completo, mas existe uma solução de programação dinâmica eficiente para digrafos acíclicos (“Longest Path -- from Wolfram MathWorld”).

Portanto, dado um Grafo Acíclico Direcionado Ponderado (DAG) e um vértice de origem s nele, encontre as maiores distâncias de s a todos os outros vértices no grafo dado.

O problema do caminho mais longo para um grafo geral não é tão fácil quanto o problema do caminho mais curto porque o problema do caminho mais longo não possui a propriedade de subestrutura ótima. De fato, o problema do caminho mais longo é NP-Difícil para um grafo geral. No entanto, o problema do caminho mais longo tem uma solução de tempo linear para grafos acíclicos direcionados. A ideia é semelhante à solução de tempo linear para o caminho mais curto em um grafo acíclico direcionado (“Longest Path in a Directed Acyclic Graph”).

Adicionalmente, de certa forma, o algoritmo do problema do caminho mais longo é semelhante ao algoritmo de Dijkstra, pois manteremos uma lista de "caminhos mais longos provisórios encontrados até agora" e marca iterativamente um deles para um vértice v como um caminho mais longo real e, em seguida, atualiza a lista com potencialmente novos caminhos mais longos combinando o caminho mais longo para v com as arestas de v . A principal diferença é que a ordenação em que essa atualização ocorrerá será escolhida no início, qualquer ordenação dos vértices compatível com a estrutura do grafo direcionada funcionará. Essas ordenações às vezes são conhecidas como "classificação topológica" (“Algorithm for Longest Paths”).

Definição

1. Problema do Caminho Longo:

Entrada: Um ‘graph’ não direcionado G = (V, E)

Saída: Sim se existir um caminho de comprimento de pelo menos |V |/4 em G, caso contrário **Não**

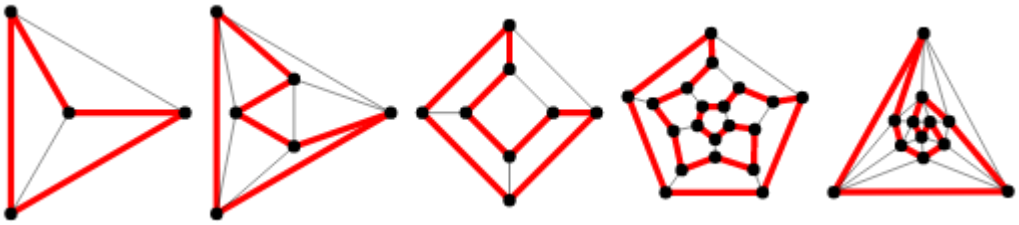
Ideia: Redução do caminho hamiltoniano. Tente pensar quantos vértices devem ser adicionado a uma instância especial G = (V, E) do problema do caminho mais longo para que, se houver é um caminho de comprimento de pelo menos |V|/4 em G então existe um caminho hamiltoniano em G.
É fácil ver que exigir que o grafo G seja conectado não diminui a dureza do nosso problema aqui. Então, no que se segue, sempre assumimos (e tentamos manter) que G é conexo (NGUYEN 1).

Ciclo Hamiltoniano

Um ciclo hamiltoniano, também chamado de circuito hamiltoniano, é um ciclo de grafo (isto é, laço fechado) através de um grafo que visita cada nó exatamente uma vez. Um grafo que possui um ciclo hamiltoniano é chamado de grafo hamiltoniano. Por convenção, o grafo singleton K_1 é considerado hamiltoniano mesmo que não possua um ciclo hamiltoniano, enquanto o grafo conectado em dois nós K2 não o é.

O ciclo hamiltoniano recebeu o nome de Sir William Rowan Hamilton, que concebeu um quebra-cabeça no qual se procurava tal caminho ao longo das bordas do poliedro de um dodecaedro (“Hamiltonian Cycle -- from Wolfram MathWorld”).

Em geral, o problema de encontrar um ciclo hamiltoniano é NP-completo, então a única maneira conhecida de determinar se um dado grafo geral tem um ciclo hamiltoniano é realizar uma busca exaustiva . Rubin (1974) descreve um procedimento de busca eficiente que pode encontrar alguns ou todos os caminhos e circuitos de Hamilton em um grafo usando deduções que reduzem bastante o retrocesso e as suposições. Um algoritmo probabilístico de Angluin e Valiant (1979), descrito por Wilf (1994), também pode ser útil para encontrar ciclos e caminhos hamiltonianos.



Todos os sólidos platônicos são hamiltonianos, como ilustrado acima (“Hamiltonian Cycle -- from Wolfram MathWorld”).

Adicionalmente, qualquer ciclo hamiltoniano pode ser convertido em um caminho hamiltoniano removendo uma de suas arestas, mas um caminho hamiltoniano pode ser estendido para um ciclo hamiltoniano somente se seus extremos forem adjacentes.

Todos os grafos hamiltonianos são biconectados, mas um grafo biconectado não precisa ser hamiltoniano (veja, por exemplo, o grafo de Petersen) (“Biconnected Graph -- from Wolfram MathWorld”).

Um grafo euleriano G (um grafo conexo no qual cada vértice tem grau par) necessariamente tem um passeio de Euler, um passeio fechado que passa por cada

aresta de G exatamente uma vez. Este passeio corresponde a um ciclo hamiltoniano no gráfico de linha L(G), de modo que o gráfico de linha de todo gráfico euleriano é hamiltoniano. Os gráficos de linha podem ter outros ciclos hamiltonianos que não correspondem aos passeios de Euler e, em particular, o gráfico de linha L(G) de cada grafo hamiltoniano G é ele próprio hamiltoniano, independentemente de o grafo G ser euleriano (R. Balakrishnan and K. Ranganathan).

Um torneio (com mais de dois vértices) é hamiltoniano se e somente se for fortemente conectado.

O número de diferentes ciclos hamiltonianos em um grafo não direcionado completo em n vértices é (n – 1)!/2 e em um grafo dirigido completo em n vértices é (n – 1)!. Essas contagens pressupõem que os ciclos que são iguais, exceto pelo ponto de partida, não são contados separadamente.

Teorema:

Se G é um grafo simples em n vértices, $n \geq 3$, e $d(v)+d(w) \geq n$ sempre que v e w não são adjacentes, então G tem um ciclo de Hamilton.

Prova. Primeiro mostramos que G é conexo. Se não, sejam v e w vértices em dois componentes conexos diferentes de G, e suponha que os componentes tenham n1 e n2 vértices. Então $d(v) \leq n1-1$ e $d(w) \leq n2-1$, então $d(v)+d(w) \leq n1+n2-2<n$.
Mas como v e w não são adjacentes, isso é uma contradição.

Agora considere um caminho possível mais longo em G: v1,v2,…,vk. Suponha, por contradição, que $k<n$, então existe algum vértice w adjacente a um de v2,v3,…,vk−1, digamos a vi. Se v1 é adjacente a vk, então w,vi,vi+1,…,vk,v1,v2,…vi−1 é um caminho de comprimento k+1, uma contradição. Portanto, v1 não é adjacente a vk, e então $d(v1)+d(vk) \geq n$. Os vizinhos de v1 estão entre {v2,v3,…,vk−1} assim como os vizinhos de vk. Considere os vértices $W=\{vl+1 \mid vl \text{ é um vizinho de } vk\}$.

Então $|N(vk)|=|W|$ e $W \subseteq \{v3,v4,\ldots,vk\}$ e $N(v1) \subseteq \{v2,v3,\ldots,vk-1\}$, então $W \cup N(v1) \subseteq \{v2,v3,\ldots,vk\}$, um conjunto com $k-1<n$ elementos. Como $|N(v1)|+|W|=|N(v1)|+|N(vk)| \geq n$, N(v1) e W devem ter um elemento comum, vj; observe que $3 \leq j \leq k-1$. Então este é um ciclo de comprimento k: v1,vj,vj+1,…,vk,vj−1,vj−2,…,v1.

Podemos renomear os vértices por conveniência: v1=w1,w2,…,wk=v2,w1.

Agora, como antes, w é adjacente a algum wl, e w,wl,wl+1,…,wk,w1,w2,…wl−1 é um caminho de comprimento k+1, uma contradição. Assim, k=n, e, renumerando os vértices por conveniência, temos um caminho de Hamilton v1,v2,…,vn. Se v1 é adjacente a vn, existe um ciclo de Hamilton, conforme desejado.

Se v1 não é adjacente a vn, os vizinhos de v1 estão entre {v2,v3,…,vn−1} assim como os vizinhos de vn. Considere os vértices $W=\{vl+1 \mid vl \text{ é um vizinho de } vn\}$.

Então $|N(vn)|=|W|$ e $W \subseteq \{v3,v4,\ldots,vn\}$, e $N(v1) \subseteq \{v2,v3,\ldots,vn-1\}$, então $W \cup N(v1) \subseteq \{v2,v3,\ldots,vn\}$, a conjunto com $n-1<n$ elementos. Como

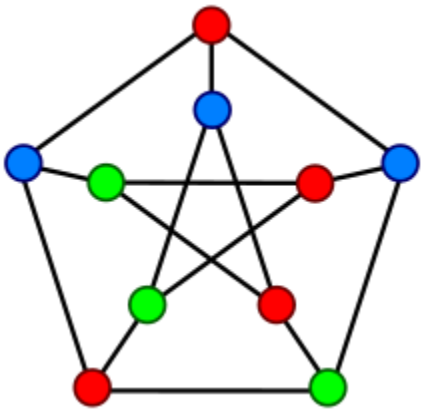
$|N(v1)|+|W|=|N(v1)|+|N(vn)|\geq n$, N(v1) e W devem ter um elemento comum, vi; observe que $3\leq i\leq n-1$. Então este é um ciclo de comprimento n: v1,vi,vi+1,…,vk,vi−1,vi−2,…,v1, e é um ciclo de Hamilton.

A propriedade usada neste teorema é chamada de propriedade Ore; se um grafo tem a propriedade Ore, ele também tem um caminho de Hamilton, mas podemos enfraquecer um pouco a condição se nosso objetivo é mostrar que existe um caminho de Hamilton. A prova deste teorema é quase idêntica à prova anterior (“Hamilton Cycles and Paths”).

Graph coloring

O problema de coloração de grafos consiste em atribuir cores a certos elementos de um grafo sujeitos a certas restrições (“Graph Coloring | Set 1 (Introduction and Applications)”). Especificamente, uma ‘coloração adequada’ de um grafo é uma atribuição de cores aos vértices do grafo de modo que dois vértices adjacentes não tenham a mesma cor (“Graph Coloring”).

Na teoria dos grafos, a coloração de grafos é um caso especial de rotulagem de grafos; é uma atribuição de rótulos tradicionalmente chamados de "cores" a elementos de um gráfico sujeitos a certas restrições. Em sua forma mais simples, é uma maneira de colorir os vértices de um grafo de modo que não haja dois vértices adjacentes da mesma cor; isso é chamado de coloração de vértice. Da mesma forma, uma coloração de aresta atribui uma cor a cada aresta de modo que não haja duas arestas adjacentes da mesma cor, e uma coloração de face de um grafo plano atribui uma cor a cada face ou região de modo que duas faces que compartilham um limite não tenham a mesma cor (Andrews and Fellows).



Normalmente, descartamos a palavra "adequada", a menos que outros tipos de coloração também estejam em discussão. É claro que as "cores" não precisam ser cores reais; eles podem ser quaisquer rótulos distintos — inteiros, por exemplo. Se um gráfico não estiver conectado, cada componente conectado pode ser

colorido independentemente; exceto onde indicado de outra forma, assumimos que os gráficos são conectados (“Graph Coloring”).

Definição:

Coloração: Uma k-coloração de um grafo G é um mapa $\varphi : V(G) \rightarrow S$ onde $|S| = k$ com a propriedade que $\varphi(u) 6= \varphi(v)$ sempre que houver uma aresta com extremidades u, v. Os elementos de S são chamados cores, e os vértices de uma cor formam uma classe de cores. O número cromático de G, denotado $\chi(G)$, é o menor inteiro k tal que G é k-colorível. Se G tem um laço, então não tem uma coloração, e definimos $\chi(G) = \infty$.

Conjunto Independente: Um conjunto de vértices é independente se eles não são adjacentes aos pares. Deixamos $\alpha(G)$ denotar o tamanho do maior conjunto independente em G. Note que em uma coloração, toda classe de cor é um conjunto independente.

Clique: Um conjunto de vértices é um clique se eles são adjacentes aos pares. Deixamos $\omega(G)$ denotar o tamanho da maior clique em G.

$\chi(G) \geq \omega(G)$
$\chi(G) \geq (|V(G)|) / (\alpha(G))$

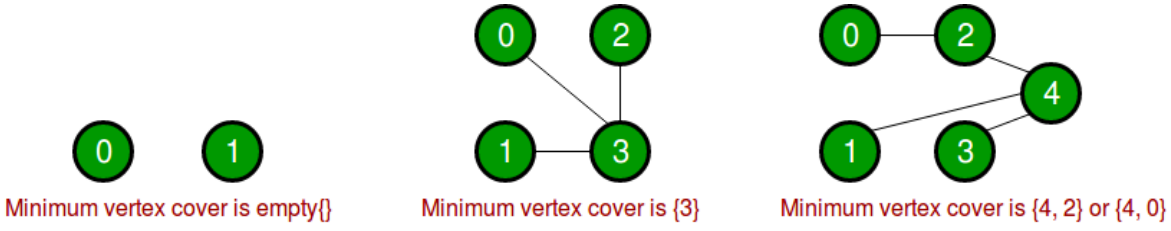
Prova: A primeira parte decorre da observação de que quaisquer dois vértices em uma clique devem receber cores diferentes. A segunda decorre da observação de que cada classe de cor em uma coloração tem tamanho $\leq \alpha(G)$ (“Graph Colouring”).

Cobertura por Vértices

Uma cobertura de vértices de um grafo G é um conjunto de vértices, Vc , tal que cada aresta em G tem pelo menos um dos vértices em Vc como ponto final. Isso significa que cada vértice no grafo está tocando pelo menos uma aresta. A cobertura de vértices é um tópico da teoria dos grafos que tem aplicações em problemas de correspondência e problemas de otimização. Uma cobertura de vértices pode ser uma boa abordagem para um problema em que todas as arestas de um grafo precisam ser incluídas na solução (“Vertex Cover”).

Adicionalmente, uma cobertura de vértices de um grafo não direcionado é um subconjunto de seus vértices tal que para cada aresta (u, v) do grafo, ou 'u' ou 'v' está na cobertura de vértices. Embora o nome seja Vertex Cover, o conjunto cobre todas as arestas do grafo fornecido. Dado um grafo não direcionado, o problema da cobertura de vértices é encontrar a cobertura de vértices de tamanho mínimo (“Vertex Cover Problem | Set 1 (Introduction and Approximate Algorithm)”).

Como nos exemplos a seguir:

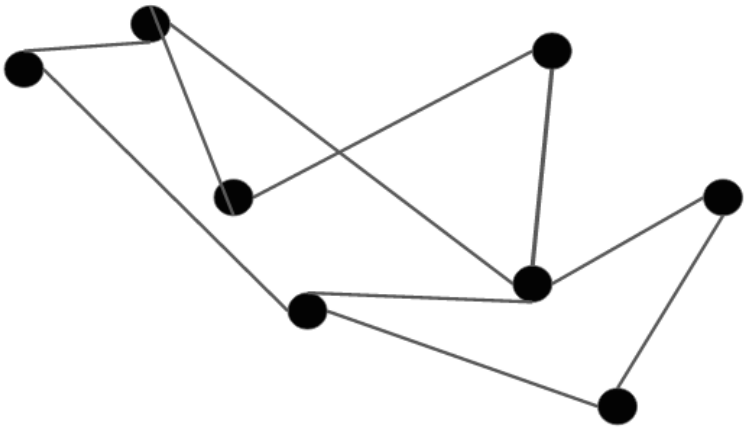


O Problema da Cobertura de Vértices é um problema NP Completo conhecido, ou seja, não há solução em tempo polinomial para isso, a menos que P = NP. Existem algoritmos de tempo polinomial aproximados para resolver o problema (“Vertex Cover Problem | Set 1 (Introduction and Approximate Algorithm)”).

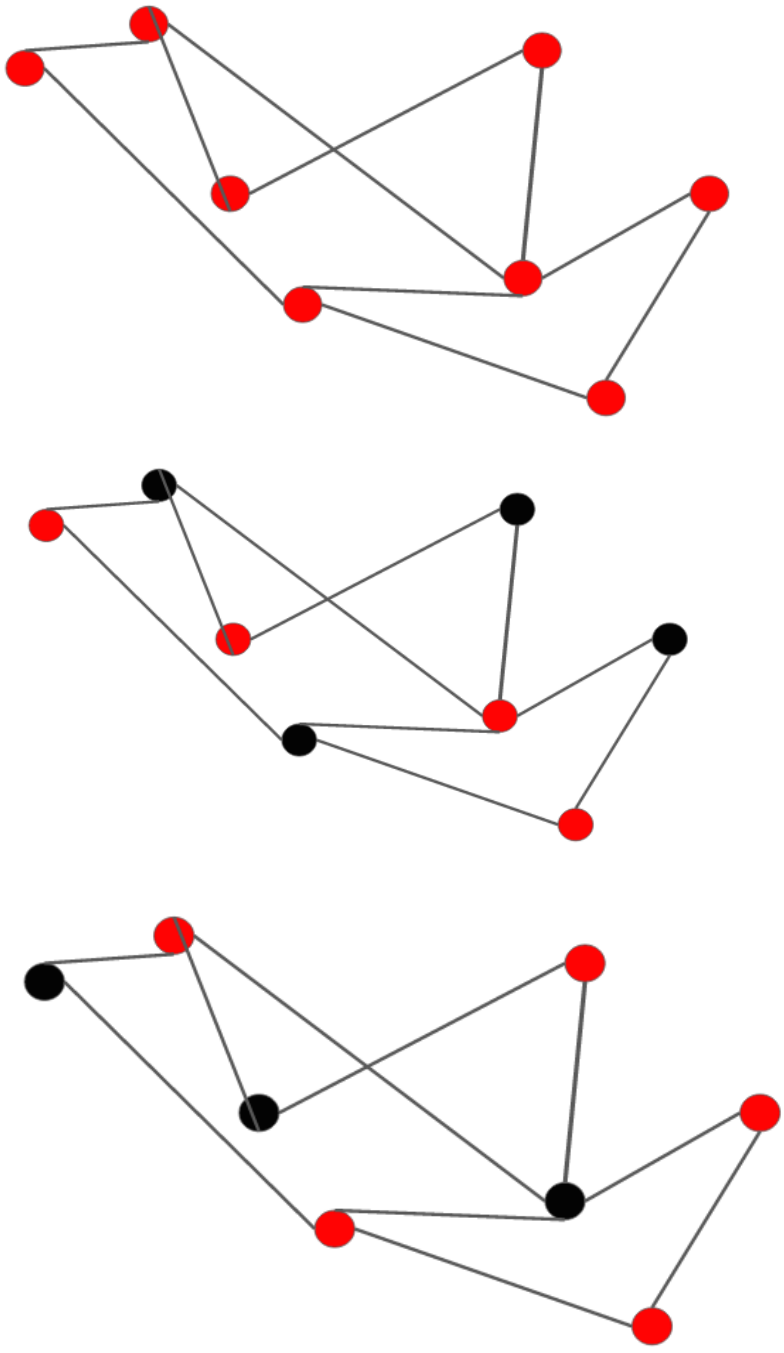
Exemplo:

Digamos que você tenha uma galeria de arte com muitos corredores e curvas. Sua galeria está exibindo pinturas muito valiosas e você deseja mantê-las seguras. Você está planejando instalar câmeras de segurança em cada corredor para que as câmeras tenham todas as pinturas à vista. Se houver uma câmera de segurança em um corredor, ela poderá ver todas as pinturas do corredor. Se houver uma câmera no canto onde dois corredores se encontram (o turno), ela pode ver pinturas em ambos os corredores. Podemos modelar este sistema como um grafo onde os nós representam os lugares onde os corredores se encontram ou quando um corredor se torna um beco sem saída, e as arestas são os corredores (“Vertex Cover”).

Este gráfico, mostra onde você colocaria as câmeras de forma que todas as pinturas fossem cobertas — esta é uma cobertura de vértice! (Existem muitas soluções).



A primeira é uma solução trivial — tenha câmeras em todos os nós. Por definição, esta é uma cobertura de vértices, pois todas as arestas do grafo devem estar conectadas a pelo menos um dos vértices da cobertura (“Vertex Cover”).



Formalmente, uma cobertura de vértice V' de um grafo não direcionado G=(V,E) é um subconjunto de V tal que $uv \in E \Rightarrow u \in V'$, ou seja, é um conjunto de vértices V' onde cada aresta tem pelo menos uma extremidade na cobertura de vértices V'. Diz-se que tal conjunto cobre as arestas de G. A figura superior mostra dois exemplos de coberturas de vértices, com algumas coberturas de vértices V' marcadas em vermelho.

Uma cobertura de vértice mínima é uma cobertura de vértice de menor tamanho possível. O número da cobertura de vértice t é o tamanho de uma cobertura de

vértice mínima, ou seja, t =|V'|. A figura inferior mostra exemplos de coberturas de vértices mínimas nos gráficos anteriores (“Vertex cover”).

ii) Ilustre a sua definição com um exemplo pequeno com dados numéricos, que seria uma instância do problema;

Problema da mochila (alocação de recursos otimizada)

Como um exemplo numérico simples, suponha que temos:

N = 3; w1 = 3, w2 = 8 e w3 = 5;
r1 = 4, r2 = 6 e r3 = 5;
e finalmente, c = 8.

Observe que dos três tipos de itens, o primeiro tipo tem o maior valor por unidade de peso. Ou seja, das três proporções,

$$\frac{r_1}{w_1} = \frac{4}{3}, \frac{r_2}{w_2} = \frac{6}{8}, \text{ and } \frac{r_3}{w_3} = \frac{5}{5},$$

A primeira razão é a maior. Portanto, parece natural tentar carregar o maior número de Type-1 possíveis na mochila. Como a capacidade da mochila é 8, tal tentativa resultará na combinação de carregamento x1 = 2, x2 = 0 e x3 = 0, com um valor total de:

$$r_1x_1 + r_2x_2 + r_3x_3 = 4 \times 2 + 6 \times 0 + 5 \times 0 = 8.$$

Essa combinação de carregamento é ideal? O fato de que esta combinação deixa uma folga desperdiçada de 2 na mochila é desconfortável. Com efeito, verifica-se que esta combinação não é ótima; e que a combinação ótima é fazer x1 = 1, x2 = 0 e x3 = 1, que atinge um valor total de 9 (“The Knapsack Problem”).

Como um exemplo alternativo, poderíamos ter o problema de pesos e lucros:
Pesos: {3, 4, 6, 5}
Lucros: {2, 3, 1, 4}
O peso da mochila é de 8 kg
O número de itens é 4
O problema acima pode ser resolvido usando o seguinte método:

xi = {1, 0, 0, 1}
= {0, 0, 0, 1}
= {0, 1, 0, 1}

O exemplo acima são as combinações possíveis. 1 indica que o item foi completamente retirado e 0 significa que nenhum item foi retirado. Como são 4 itens, as combinações possíveis serão:

24 = 16; Então, existem 16 combinações possíveis que podem ser feitas usando o problema acima. Uma vez que todas as combinações são feitas, temos que selecionar a combinação que fornece o lucro máximo (“DAA | 0/1 Knapsack Problem - javatpoint”).

Caminho mais Longo/ ou Crítico

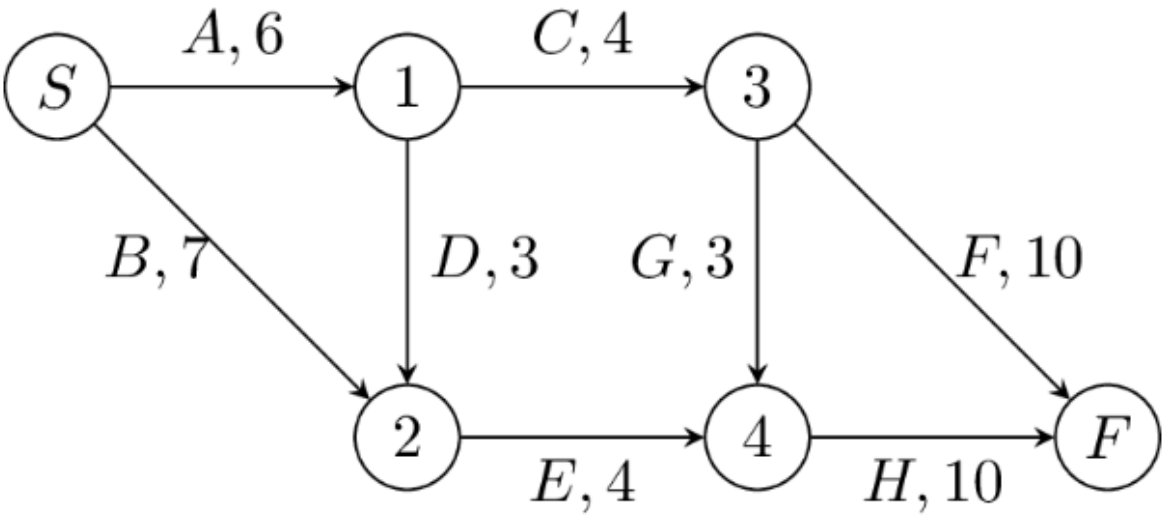
Considere a tabela a seguir, listando as tarefas A − H , o tempo esperado de conclusão de cada tarefa e as tarefas necessárias antes que uma determinada tarefa possa ser iniciada (“Algorithm for Longest Paths”).

Tabela mostrando os pré-requisitos para cada tarefa:

Task	Time	Prerequisites
A	6	
B	7	
C	4	A
D	3	A
E	4	B,D
F	10	C
G	3	C
H	10	E,G

2. Ilustração Caminho mais Longo/ ou Crítico (“Algorithm for Longest Paths”).

Aqui está o gráfico correspondente que codifica esta informação:



3. Ilustração Caminho mais Longo/ ou Crítico (“Algorithm for Longest Paths”).

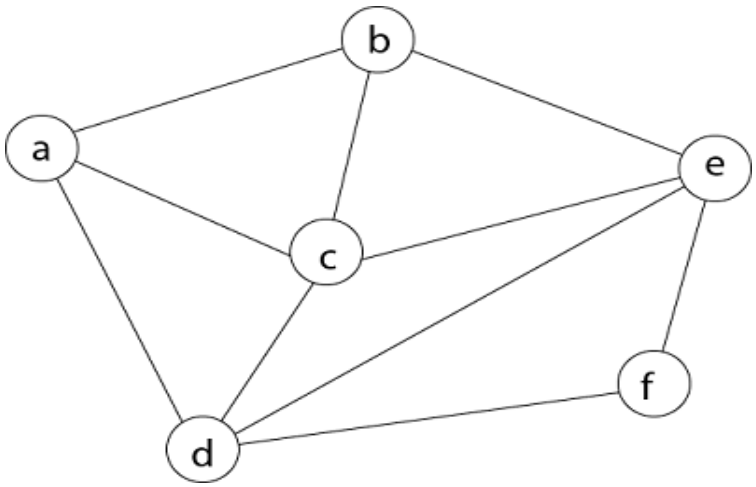
Descrevemos como o gráfico acima foi construído. Fazemos um vértice para o início, um vértice para o final e depois outro vértice para cada conjunto de dependências, ou seja, as entradas da terceira coluna. Em seguida, desenhamos uma aresta para cada letra, começando no vértice correspondente ao seu conjunto de pré-requisitos (ou no início, se não houver nenhum), e terminando no vértice que o contém como pré-requisito (ou no final, se nenhuma tarefa exigir como pré-requisito) (“Algorithm for Longest Paths”).

O algoritmo de caminho mais longo no gráfico de exemplo. O vértice inicial é S , então $\ell (S) = 0$.
O vértice 1 tem apenas uma aresta de entrada: A , com peso 6, então $\ell (1) = 6 + \ell (S) = 6$.
O vértice 2 tem duas arestas de entrada: B e D , e assim vemos que $\ell (2)$ é o máximo de $w (D) + \ell (1) = 3 + 6 = 9$, e $w (B) + \ell (S) = 7 + 0 = 7$, então $\ell (2) = 9$.
O vértice 3 tem apenas uma aresta de entrada: C , então $\ell (3) = w (C) + \ell (1) = 4 + 6 = 10$.
O vértice 4 tem duas arestas de entrada: G e E , então $\ell (4)$ é o máximo de $w (G) + \ell (3) = 3 + 10 = 13$ e $w (E) + \ell (2) = 4 + 9 = 13$. Assim, o máximo é alcançado de duas maneiras diferentes, e vemos que existem dois caminhos de comprimento 13 de S a 4 -- $S - 1 - 3 - 4$ e $S - 1 - 2 - 4$.
Finalmente, o vértice F tem duas arestas de entrada, F e H , e então $\ell (F)$ é o máximo de $w (F) + \ell (3) = 10 + 10 = 20$ e $w (H) + \ell (4) = 10 + 13 = 23$. Existem dois caminhos que atingem esse máximo -- $A - C - G - H$ e $A - D - E - H$ (“Algorithm for Longest Paths”).

Ciclo Hamiltoniano

Dado um grafo G = (V, E) temos que encontrar o Circuito Hamiltoniano usando a abordagem Backtracking. Começamos nossa busca a partir de qualquer vértice arbitrário, digamos 'a'. Este vértice 'a' se torna a raiz de nossa árvore implícita. O primeiro elemento de nossa solução parcial é o primeiro vértice intermediário do Ciclo Hamiltoniano a ser construído. O próximo vértice adjacente é selecionado por ordem alfabética. Se em qualquer estágio qualquer vértice arbitrário faz um ciclo com qualquer vértice diferente do vértice 'a' então dizemos que o beco sem saída foi alcançado. Nesse caso, retrocedemos um passo e, novamente, a busca começa selecionando outro vértice e retrocedemos o elemento da parcial; solução deve ser removida. A busca usando backtracking é bem sucedida se um ciclo hamiltoniano for obtido.

Exemplo: Considere um grafo G = (V, E) mostrado na fig. temos que encontrar um circuito hamiltoniano usando o método Backtracking.



Solução: Primeiramente, iniciamos nossa busca com o vértice 'a.' este vértice 'a' torna-se a raiz da nossa árvore implícita.

Em seguida, escolhemos o vértice 'b' adjacente a 'a', pois vem primeiro na ordem lexicográfica (b, c, d).

Em seguida, selecionamos 'c' adjacente a 'b'.

Em seguida, selecionamos 'd' adjacente a 'c'.

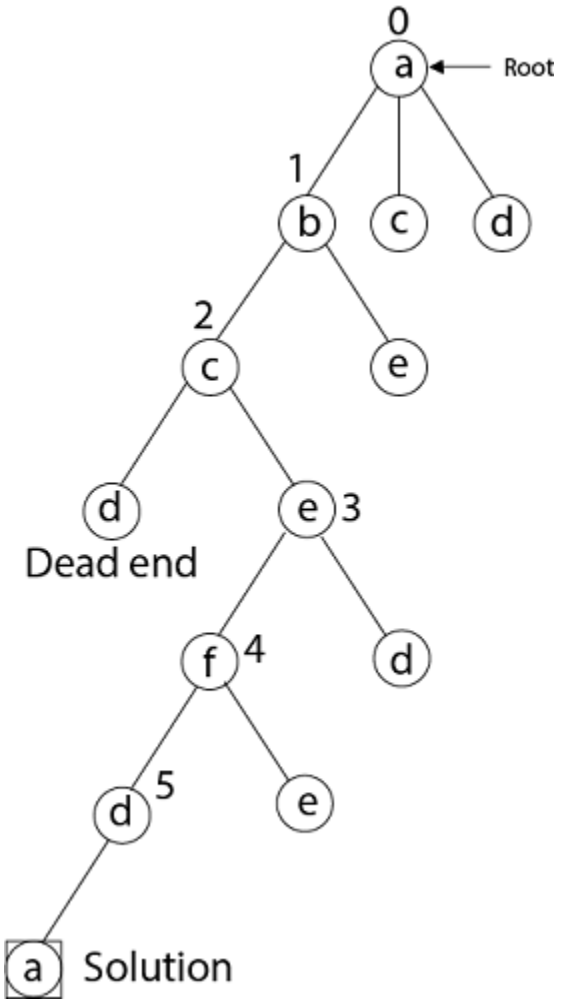
Em seguida, selecionamos 'e' adjacente a 'd'.

Em seguida, selecionamos o vértice 'f' adjacente a 'e'. O vértice adjacente a 'f' é d e e, mas eles já o visitaram. Assim, chegamos ao beco sem saída, retrocedemos um passo e removemos o vértice 'f' da solução parcial.

A partir do retrocesso, o vértice adjacente a 'e' é b, c, d, e f a partir do qual o vértice 'f' já foi verificado e b, c, d já foram visitados. Então, novamente voltamos um passo. Agora, os vértices adjacentes a d são e, f a partir do qual e já foi verificado, e

adjacentes a 'f' são d e e. Se o vértice 'e', revisitado, obtemos um estado morto. Então, novamente, retrocedemos um passo.

Agora, adjacente a c é 'e' e adjacente a 'e' é 'f' e adjacente a 'f' é 'd' e adjacente a 'd' é 'a.' Aqui, obtemos o Ciclo Hamiltoniano, pois todos os vértices, exceto o vértice inicial 'a', são visitados apenas uma vez. (a - b - c - e - f -d - a).

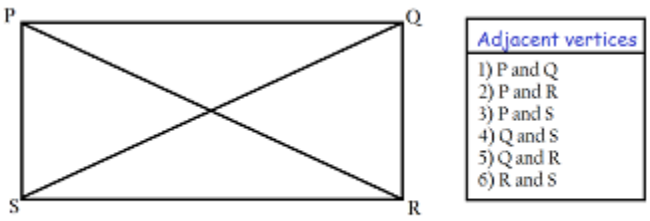


Aqui geramos um circuito hamiltoniano, mas outro circuito hamiltoniano também pode ser obtido considerando outro vértice (“Hamiltonian Circuit Problems - javatpoint”).

Graph coloring

Um grafo é uma coleção de vértices que estão conectados uns aos outros por meio de arestas. Os vértices são ditos adjacentes se eles estão próximos um do outro e uma aresta passa entre eles (“Coloring & Traversing Graphs in Discrete Math”).

A imagem abaixo mostra um grafo com seus vértices adjacentes.



Colorir um grafo nada mais é do que atribuir uma cor a cada vértice de um grafo, garantindo que os vértices adjacentes não recebam a mesma cor.

Colorindo o gráfico mostrado acima na acima. Começaremos usando o vértice P.

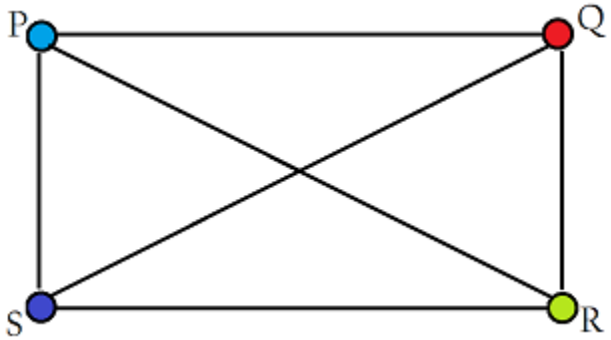
Vértice P: Vamos colorir este vértice de azul.

Vértice Q: Como Q é adjacente a P, não podemos atribuir azul a ele. Vamos colori-lo de vermelho.

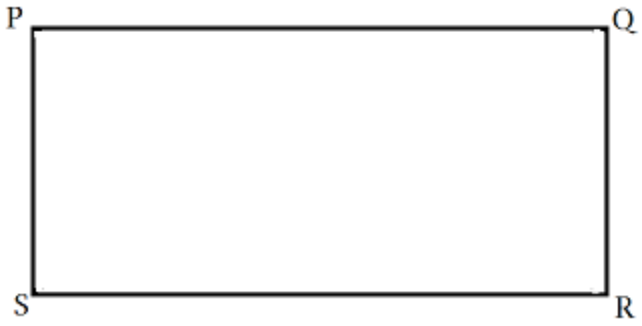
Vértice R: Como o vértice R é adjacente a Q e P, não podemos atribuir azul ou vermelho a ele. Vamos colori-lo de verde.

Vértice S: Como S é adjacente a P, Q e R, não podemos atribuir a ele nenhuma das cores anteriores. Vamos colorir de roxo.

O gráfico colorido seria algo como a figura abaixo:



Usamos um total de quatro cores para colorir o gráfico. Isso nem sempre é o caso com todos os gráficos que você encontraria, no entanto. Abaixo descreve outro exemplo.



Vamos colorir o gráfico PQRS da acima, começando pelo vértice P.

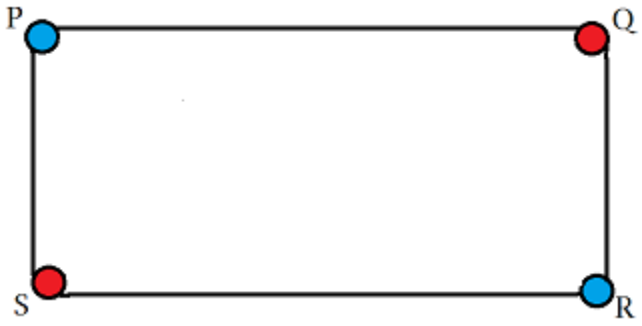
Vértice P: Vamos atribuir a cor azul.

Vértice Q: Q é adjacente a P. Não podemos atribuir a mesma cor que P. Vamos colori-lo de vermelho.

Vértice R: R é adjacente a Q. Não podemos atribuir a mesma cor que Q. Vamos colori-lo de azul.

Vértice S: S é adjacente a P. Não podemos atribuir a mesma cor que R. Vamos colori-lo de vermelho.

O gráfico colorido seria semelhante ao da figura abaixo:



Neste exemplo, apenas duas cores foram suficientes para colorir o gráfico ("Coloring & Traversing Graphs in Discrete Math").

Cobertura por Vértices

Apresentamos agora uma descrição formal do algoritmo. Isto é seguido por um pequeno exemplo que ilustra os passos do algoritmo. Começamos definindo dois procedimentos.

Procedimento: Dado um grafo simples G com n vértices e uma cobertura de vértices C de G, se C não tiver vértices removíveis, imprima C. Caso contrário, para cada vértice removível v de C, encontre o número $\rho(C-\{v\})$ de vértices

removíveis da cobertura de vértices C−{v}. Seja vmax um vértice removível tal que ρ(C−{vmax}) seja um máximo e obtenha a cobertura de vértice C−{vmax}. Repita até que a tampa do vértice não tenha vértices removíveis.

Procedimento: Dado um grafo simples G com n vértices e uma cobertura mínima de vértices C de G, se não houver nenhum vértice v em C tal que v tenha exatamente um vizinho w fora de C, produza C. Caso contrário, encontre um vértice v em C tal que v tem exatamente um vizinho w fora de C. Defina Cv,w removendo v de C e adicionando w a C. Execute o procedimento 3.1 em Cv,w e produza a cobertura de vértices resultante.

Algoritmo: Dado como entrada um grafo simples G com n vértices rotulados 1, 2, ..., n, procure uma cobertura de vértices de tamanho no máximo k. Em cada estágio, se a cobertura de vértices obtida tiver tamanho no máximo k, então pare.

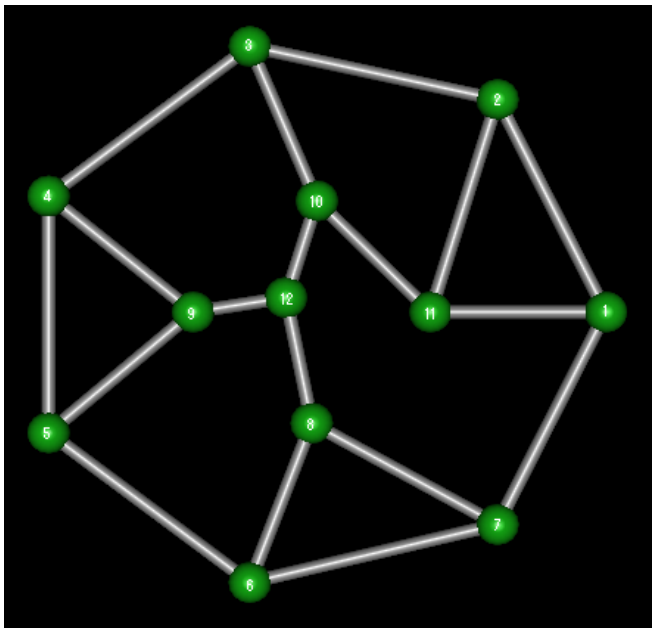
Parte I. Para i = 1, 2, ..., n por sua vez
Inicialize a cobertura de vértice Ci = V−{i}.
Execute o procedimento 3.1 em Ci.
Para r = 1, 2, ..., n−k execute o procedimento 3.2 repetido r vezes.
O resultado é uma cobertura de vértice mínima Ci.

Parte II. Para cada par de vértices mínimos cobre Ci, Cj encontrados na Parte I
Inicialize a cobertura de vértice Ci, j = Ci ∪ Cj .
Execute o procedimento 3.1 em Ci, j.
Para r = 1, 2, ..., n−k execute o procedimento 3.2 repetido r vezes.

O O resultado é uma cobertura de vértice mínima Ci, j.

Demonstramos as etapas do algoritmo com um pequeno exemplo. A entrada é o gráfico Frucht [2] mostrado abaixo com n = 12 vértices rotulados

V = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11, 12} (Dharwadker).



Procuramos uma cobertura de vértices de tamanho no máximo k = 7. Parte I para i = 1 e i = 2 produz coberturas de vértices C1 e C2 de tamanho 8, então damos os detalhes a partir de i = 3. Inicializamos a cobertura de vértices Como

C3 = V−{3}={1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12}.

Realizamos agora o procedimento 3.1. Aqui estão os resultados em forma de tabela:

Cobertura de vértice C3 = {1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12}. Tamanho: 11.

O algoritmo completo se encontra neste site.

iii) Pesquise uma expressão para o número de soluções possíveis para esse problema, que deve ser uma função de crescimento exponencial e ilustre no seu exemplo;

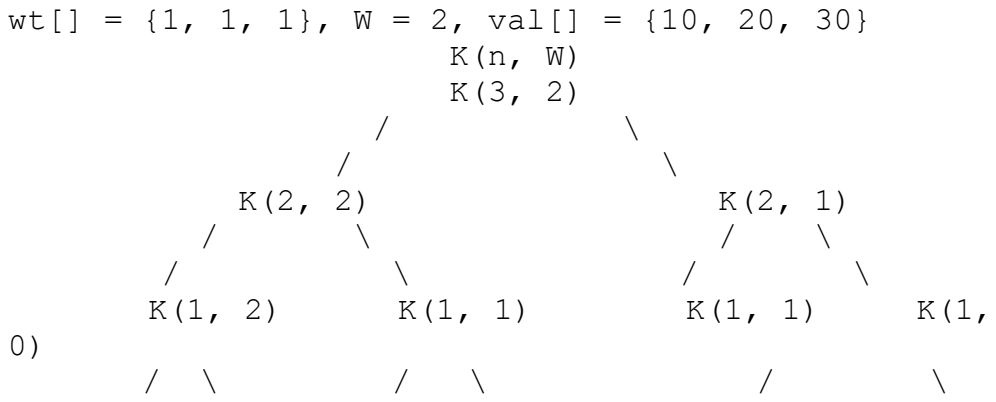
Problema da mochila (alocação de recursos otimizada)

O Problema da mochila é NP-difícil, então não conhecemos um algoritmo de tempo polinomial para isso. No entanto, ele possui um algoritmo de tempo pseudo polinomial que podemos usar para mochila. Este algoritmo usa programação dinâmica para encontrar a solução ótima (Katherine Lai).

A seguir é apresentado uma possível solução com uma função de crescimento exponencial e links adicionais:

Veja a seguinte árvore de recursão, K(1, 1) está sendo avaliada duas vezes. A complexidade de tempo desta solução recursiva ingênua é exponencial (2^n).

Na árvore de recursão a seguir, K() refere-se a knapSack(). Os dois parâmetros indicados na árvore de recursão a seguir são ne W. A árvore de recursão é para as seguintes entradas de amostra (“0-1 Knapsack Problem | DP-10”) .



```
      /       \      /        \            /              \
K(0, 2)   K(0, 1)   K(0, 1)   K(0, 0)   K(0, 1)    K(0, 0)
```

Árvore de recursão para capacidade de mochila 2 unidades e 3 itens de 1 unidade de peso.

- Análise de Complexidade:
  Complexidade de tempo: O(2n).
  Como existem subproblemas redundantes.

- Espaço Auxiliar :O(1) + O(N).
  Como nenhuma estrutura de dados extra foi usada para armazenar valores, mas o espaço de pilha auxiliar O(N) (ASS) foi usado para pilha de recursão (“0-1 Knapsack Problem | DP-10”).

O código para o problema apresentado acima se encontra neste link.
Como referência adicionais deixo o artigo: O número de soluções viáveis para um problema da mochila (The Number of Feasible Solutions to a Knapsack Problem)

Caminho mais Longo/ ou Crítico

O problema do caminho mais longo é o problema de encontrar um caminho de comprimento máximo em um grafo. As soluções polinomiais para este problema são conhecidas apenas para pequenas classes de grafos, enquanto é NP-difícil em grafos gerais, pois é uma generalização do problema do caminho hamiltoniano (“The Longest Path Problem has a Polynomial Solution on Interval Graphs”).

Como referência adicionais deixo o artigo: The Longest Path Problem: an NP-Complete Example. Pois contem ótimos códigos como exemplo.

Ciclo Hamiltoniano

Graph coloring

Cobertura por Vértices

iv) Pesquise e descreva possíveis aplicações práticas para cada problema.

Problema da mochila (alocação de recursos otimizada)

Nesse escopo, com o avanço da tecnologia problemas como o Descarregamento de computação multiusuário como problema de mochila múltipla para computação de borda móvel 5G (Multi-user computation offloading as Multiple Knapsack Problem for 5G Mobile Edge Computing) surge e para sua solução e usado o problema da mochila.

O Mobile Edge Computing (MEC) é um conceito emergente em tecnologia de rede que visa empurrar a computação recursos mais próximos dos usuários, na borda da rede, visando uma melhoria nos serviços de nuvem tradicionais por melhor distribuição, dimensionamento mais fácil e acesso mais rápido. A grosso modo, O equipamento do usuário (UE) pode se conectar a um ou mais MEC Servidores (MECS) para descarregar tarefas que seriam muito lentas ou muito caras em termos de energia da bateria (István Ketykó and László Kecskés 1). Os pesquisadores fizeram um modelo geral do sistema e, em seguida, investigamos um problema com a omissão de alguns parâmetros, portanto, acabaram no MKP. Os resultados da simulação demonstram que as configurações, mesmo com parâmetros pequenos, têm instâncias com tempos de execução longos pelos solucionadores exatos, e é por isso que as abordagens heurísticas são necessárias, portanto, há um potencial real em uma abordagem teórica de jogos distribuída (István Ketykó and László Kecskés 5).

Apesar do uso de MKP eu, autor deste trabalho, escolhi apresentar esse artigo por se tratar de uma aplicação recente de uma variação interessante do problema da mochila.

Caminho mais Longo/ ou Crítico

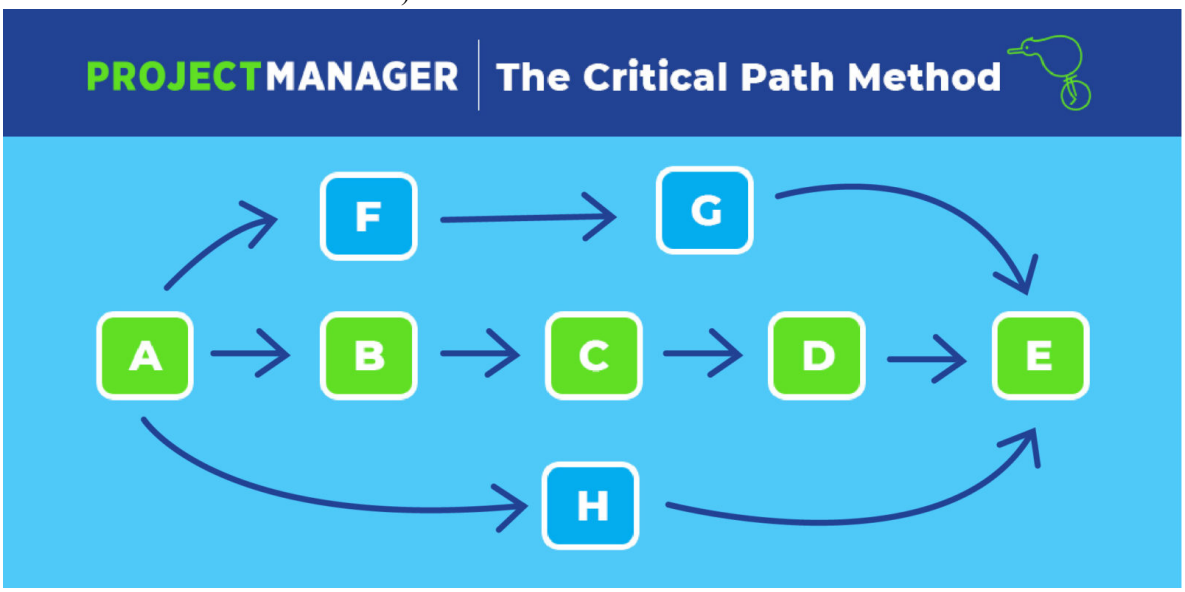
No gerenciamento de projetos, o caminho crítico é a sequência mais longa de tarefas que devem ser concluídas para concluir um projeto. As tarefas no caminho crítico são chamadas de atividades críticas porque, se estiverem atrasadas, toda a conclusão do projeto será atrasada (“Critical Path Method: The Ultimate Guide to Critical Path”).

Encontrar o caminho crítico é muito importante para os gerentes de projeto porque permite:

- Estimar com precisão a duração total do projeto;
- Identificar dependências de tarefas, restrições de recursos e riscos do projeto;
- Priorizar tarefas e criar cronogramas de projeto realistas.

Para encontrar o caminho crítico, os gerentes de projeto usam o caminho crítico algoritmo de método (CPM) para definir a menor quantidade de tempo necessária para concluir cada tarefa com a menor quantidade de folga.

A seguir um exemplo de um diagrama CPM. Embora seja de alto nível, pode ajudá-lo a visualizar o significado de um caminho crítico para um cronograma de projeto. Por enquanto, usaremos este diagrama de caminho crítico para explicar os elementos que compõem o método CPM (“Critical Path Method: The Ultimate Guide to Critical Path”).

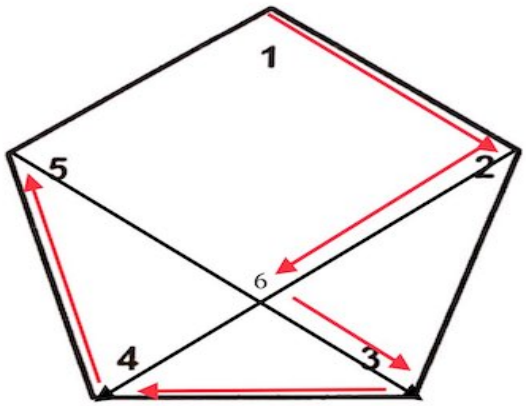


4. Ilustração Caminho mais Longo/ ou Crítico (“Critical Path Method: The Ultimate Guide to Critical Path”).

Como ilustrado no diagrama de caminho crítico, as atividades do projeto são representadas por letras e o caminho crítico é destacado em verde. As tarefas F, G e H são atividades não críticas com folga ou folga. Também podemos identificar dependências de tarefas entre as atividades do caminho crítico, e também entre as atividades (A, F e G) ou (A, H e E), que são tarefas paralelas (“Critical Path Method: The Ultimate Guide to Critical Path”).
De exemplo adicional, disponibilizo o do caminho crítico da Harvard Business Review, que mostra um cronograma de caminho crítico para a construção de uma casa.

Ciclo Hamiltoniano

Um motorista de entrega quer fazer o melhor uso de seu tempo e quilometragem do veículo planejando uma rota que exigirá a menor quantidade de quilometragem e tempo. na figura abaixo, as casas para as quais ela planeja fazer entregas estão numeradas de dois a seis, com o armazém de entrega marcado como um. Para reduzir o tempo que ela gasta dirigindo e entregando, o motorista planeja sua rota com as paradas na seguinte ordem: 1, 2, 6, 3, 4, 5 (“Hamiltonian Circuit, Path & Examples | What is a Hamiltonian Circuit? - Video & Lesson Transcript”).



Ao seguir este trajeto na figura acima, fica claro que cada uma dessas paradas é percorrida apenas uma vez pelo motorista. Como é feita apenas uma visita a cada uma dessas paradas, o caminho escolhido pelo motorista representa um caminho hamiltoniano.

O que é um caminho hamiltoniano? Um caminho hamiltoniano, bem como sua contraparte, o circuito hamiltoniano, representa um componente da teoria dos grafos. Na teoria dos grafos, um grafo é uma representação visual de dados caracterizada pela presença de nós ou vértices. Na Figura 1, cada uma das paradas no caminho do driver constitui os nós ou vértices.

O segundo componente de um grafo é uma aresta, ou uma linha, que conecta dois nós. Por exemplo, na figura acima, a linha entre os nós 1 e 2 representa uma aresta. Essas linhas, ou arestas, servem para marcar a relação entre os nós em um grafo. Uma coisa importante a notar é que ao formar um caminho hamiltoniano em um grafo, nem todas as arestas precisam ser usadas. No entanto, todos os vértices presentes em um grafo devem ser visitados apenas uma vez ao percorrer o caminho.

Além de servir como um passatempo divertido quando alguém está entediado e precisa de algo intelectualmente estimulante para fazer, desenvolver um caminho ou circuito hamiltoniano, como será discutido na próxima seção, tem uma variedade de outros usos. Algumas dessas aplicações do mundo real incluem ("Hamiltonian Circuit, Path & Examples | What is a Hamiltonian Circuit? - Video & Lesson Transcript"):

- Planejamento urbano: caminhos e circuitos hamiltonianos podem ajudar no planejamento de estradas e outras infraestruturas em ambientes urbanos.
- Redes sociais: Da mesma maneira, os vértices, ou nós, em um caminho ou circuito hamiltoniano podem ser usados para refletir os tipos de atividades sociais presentes nas redes e sites de mídia social.

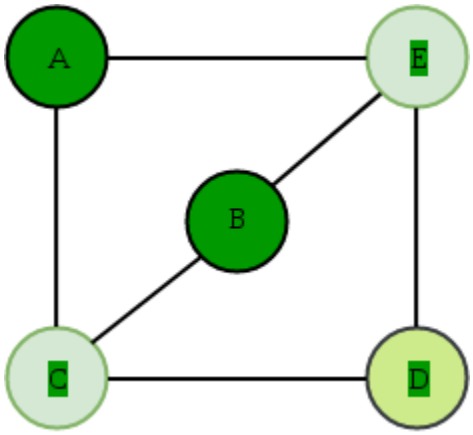
- Sistemas GPS: Caminhos e circuitos hamiltonianos são usados regularmente em tecnologias de mapas que ajudam a encontrar a rota mais curta entre diferentes locais.
- Química e física: caminhos e circuitos hamiltonianos também podem ser úteis ao estudar a interação entre moléculas.
- Engenharia elétrica e ciência da computação: Os pesquisadores dessas áreas também fazem uso de caminhos e circuitos hamiltonianos para criar placas de circuito para computadores e outros dispositivos.

Graph coloring

Como já visto, o problema de coloração de grafos consiste em atribuir cores a certos elementos de um grafo sujeitos a certas restrições.

A coloração de vértices é o problema de coloração de grafos mais comum. O problema é, dadas m cores, encontrar uma maneira de colorir os vértices de um grafo de modo que dois vértices adjacentes não sejam coloridos usando a mesma cor. Os outros problemas de coloração de grafos como Coloração de Arestas (Nenhum vértice é incidente a duas arestas da mesma cor) e Coloração de Faces (Coloração de Mapas Geográficos) podem ser transformados em coloração de vértices ("Graph Coloring | Set 1 (Introduction and Applications)").

Número cromático: O menor número de cores necessárias para colorir um grafo G é chamado de número cromático. Por exemplo, o seguinte pode ser colorido com um mínimo de 2 cores.



1) Cronograma ou agendamento: Suponha que queremos fazer um cronograma de exames para uma universidade. Listamos diferentes disciplinas e alunos matriculados em cada disciplina. Muitas disciplinas teriam alunos comuns (do mesmo lote, alguns alunos atrasados, etc). Como agendamos o exame para que não sejam agendados dois exames com um aluno em comum ao mesmo tempo? Quantos horários mínimos são necessários para agendar todos os exames? Este problema pode ser representado como um grafo onde cada vértice é um sujeito e uma aresta entre dois vértices significa que há um aluno comum. Portanto, este é

um problema de coloração de grafos onde o número mínimo de intervalos de tempo é igual ao número cromático do grafo.

2) Atribuição de Radiofrequência Móvel: Quando as frequências são atribuídas a torres, as frequências atribuídas a todas as torres no mesmo local devem ser diferentes. Como atribuir frequências com essa restrição? Qual é o número mínimo de frequências necessárias? Este problema também é uma instância do problema de coloração de grafos onde cada torre representa um vértice e uma aresta entre duas torres representa que elas estão ao alcance uma da outra.

3) Sudoku: Sudoku também é uma variação do problema de coloração de grafos onde cada célula representa um vértice. Existe uma aresta entre dois vértices se eles estiverem na mesma linha ou na mesma coluna ou no mesmo bloco.

4) Alocação de registradores: Na otimização do compilador, a alocação de registradores é o processo de atribuir um grande número de variáveis de programa alvo em um pequeno número de registradores da CPU. Este problema também é um problema de coloração de grafos.

5) Gráficos Bipartidos: Podemos verificar se um grafo é Bipartido ou não colorindo o gráfico com duas cores. Se um dado gráfico é bicolor, então é bipartido, caso contrário não. Veja isto para mais detalhes.

6) Coloração do mapa: mapas geográficos de países ou estados onde duas cidades adjacentes não podem ser atribuídas da mesma cor. Quatro cores são suficientes para colorir qualquer mapa (ver Teorema das Quatro Cores) (“Graph Coloring | Set 1 (Introduction and Applications)”).

Cobertura por Vértices

Como apresentado, a cobertura de vértices é um tópico da teoria dos grafos que tem aplicações em problemas de correspondência e problemas de otimização. Uma cobertura de vértices pode ser uma boa abordagem para um problema em que todas as arestas de um grafo precisam ser incluídas na solução (“Vertex Cover”). Mais exemplos seriam como, um estabelecimento comercial interessado em instalar o menor número possível de câmeras de circuito fechado cobrindo todos os corredores (bordas) conectando todos os cômodos (nós) em um piso pode modelar o objetivo como um problema de minimização de cobertura de vértices. O problema também tem sido usado para modelar a eliminação de sequências de DNA repetitivas para aplicações de biologia sintética e engenharia metabólica (“Vertex cover”).

Referências

"Algorithm for Longest Paths." *3.5 Algorithm for Longest Paths*, https://ptwiddle.github.io/Graph-Theory-Notes/s_graphalgorithms_longest-paths.html. Accessed 17 September 2022.

Andrews, Jim, and Mike Fellows. "Graph coloring." *Wikipedia*, https://en.wikipedia.org/wiki/Graph_coloring. Accessed 19 September 2022.

"Biconnected Graph -- from Wolfram MathWorld." *Wolfram MathWorld*, https://mathworld.wolfram.com/BiconnectedGraph.html. Accessed 19 September 2022.

"Coloring & Traversing Graphs in Discrete Math." *Study.com*, https://study.com/academy/lesson/coloring-traversing-graphs-in-discrete-math.html. Accessed 19 September 2022.

"Critical Path Method: The Ultimate Guide to Critical Path." *Project Management Software*, https://www.projectmanager.com/guides/critical-path-method. Accessed 17 September 2022.

"DAA | 0/1 Knapsack Problem - javatpoint." *Javatpoint*, https://www.javatpoint.com/0-1-knapsack-problem. Accessed 19 September 2022.

Dharwadker, Ashay. "The Vertex Cover Algorithm." *Four Color Theorem*, https://www.dharwadker.org/vertex_cover/. Accessed 19 September 2022.

"Graph Coloring." *5.8 Graph Coloring*, https://www.whitman.edu/mathematics/cgt_online/book/section05.08.html. Accessed 19 September 2022.

"Graph Coloring | Set 1 (Introduction and Applications)." *GeeksforGeeks*, 10 November 2020, https://www.geeksforgeeks.org/graph-coloring-applications/. Accessed 19 September 2022.

"Graph Colouring." *6 Graph Colouring*, https://www.sfu.ca/~mdevos/notes/graph/445_colouring0.pdf. Accessed 19 September 2022.

"Hamilton Cycles and Paths." *5.3 Hamilton Cycles and Paths*, https://www.whitman.edu/mathematics/cgt_online/book/section05.03.html. Accessed 19 September 2022.

"Hamiltonian Circuit, Path & Examples | What is a Hamiltonian Circuit? - Video & Lesson Transcript." *Study.com*, 27 June 2022, https://study.com/learn/lesson/hamiltonian-circuit-path-examples.html. Accessed 19 September 2022.

"Hamiltonian Circuit Problems - javatpoint." *Javatpoint*, https://www.javatpoint.com/hamiltonian-circuit-problems. Accessed 19 September 2022.

"Hamiltonian Cycle -- from Wolfram MathWorld." *Wolfram MathWorld*, https://mathworld.wolfram.com/HamiltonianCycle.html. Accessed 19 September 2022.

István Ketykó, and László Kecskés. "Multi-user computation offloading as Multiple Knapsack Problem for 5G Mobile Edge Computing." *IEEE*, 2016, p. 5, https://ieeexplore.ieee.org/document/7561037. Accessed 11 09 2022.

Katherine Lai. "The Knapsack Problem and Fully Polynomial Time Approximation Schemes (FPTAS)." *The Knapsack Problem and Fully Polynomial Time Approximation Schemes (FPTAS)*, 10 March 2006, https://math.mit.edu/~goemans/18434S06/knapsack-katherine.pdf. Accessed 11 September 2022.

"Knapsack problem." *Wikipedia*, https://en.wikipedia.org/wiki/Knapsack_problem. Accessed 11 September 2022.

"The Knapsack Problem." *The Knapsack Problem*, https://personal.utdallas.edu/~scniu/OPRE-6201/documents/DP3-Knapsack.pdf. Accessed 11 September 2022.

Koroth, Unni Krishnan. "The NP Complete Problems — Why we have Self driving cars, Colonization plans for Mars, but no true…." *Student Voices*, 27 November 2016, https://mystudentvoices.com/the-np-complete-problems-why-we-have-self-driving-cars-colonization-plans-for-mars-but-no-true-b907660ee163. Accessed 11 September 2022.

"Longest Path -- from Wolfram MathWorld." *Wolfram MathWorld*, https://mathworld.wolfram.com/LongestPath.html. Accessed 17 September 2022.

"Longest Path in a Directed Acyclic Graph." *GeeksforGeeks*, 23 June 2022, https://www.geeksforgeeks.org/find-longest-path-directed-acyclic-graph/. Accessed 17 September 2022.

"The Longest Path Problem has a Polynomial Solution on Interval Graphs." *George B. Mertzios*, 12 May 2010, https://mertzios.net/papers/Jour/Jour_Longest-Interval.pdf. Accessed 17 September 2022.

NGUYEN, THINH D. "A SPECIAL CASE OF LONGEST PATH PROBLEM." https://osf.io/gn4vs/download.

R. Balakrishnan, and K. Ranganathan. "A Textbook of Graph Theory."

"Vertex Cover." *Brilliant*, https://brilliant.org/wiki/vertex-cover/. Accessed 19 September 2022.

"Vertex cover." *Wikipedia*, https://en.wikipedia.org/wiki/Vertex_cover. Accessed 19 September 2022.

"Vertex cover." *Wikiwand*, https://www.wikiwand.com/en/Vertex_cover. Accessed 19 September 2022.

"Vertex Cover Problem | Set 1 (Introduction and Approximate Algorithm)." *GeeksforGeeks*, 18 August 2021,

https://www.geeksforgeeks.org/vertex-cover-problem-set-1-introduction-approximate-algorithm-2/. Accessed 19 September 2022.

"What is the Knapsack Problem? - Definition from Techopedia." *Techopedia*, https://www.techopedia.com/definition/20272/knapsack-problem. Accessed 11 September 2022.

"0-1 Knapsack Problem | DP-10." *GeeksforGeeks*, 2 June 2022, https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/. Accessed 11 September 2022.