UNIVERSIDADE FEDERAL DE RORAIMA
PROFESSOR (A) : Herbert Oliveira Rocha
DISCIPLINA DE ANALISE DE ALGORITMOS.
ALUNO : Ibukun Chife didier Adjitche

Resolução da Lista 2.

[QUESTÃO – 01]

Especifique cada problema e calcule o M.C. (melhor caso), P.C. (pior caso), C.M. (caso médio) e a ordem de complexidade para algoritmos (os melhores existentes e versão recursiva e não-recursiva) para problemas abaixo. Procure ainda, pelo L.I. (Limite Inferior) de tais problemas:

(A) N-ésimo número da seqüência de Fibonacci

(B) Geração de todas as permutações de um número

[RESOLUÇÃO – 01]

(A) N-ésimo número da seqüência de Fibonacci

Pode ser encontrado com a função recursiva com complexidade de 2^n:

fib(n)
if(n >=1)
return 1
else
fib(n-1) + fib(n-2)
Cuja análise assintótica gera a equação T(n) = T(n-1) + T(n-2) + c (uma constante).

O limite inferior pode ser calculado aproximando o valor de F(n-1) a F(n-2).
T(n) = T(n-1) + T(n-2) + c
= 2T(n-2) + c
= 2*(2T(n-4) + c) + c
= 4T(n-4) + 3c
= 8T(n-8) + 7c
= 2^k * T(n - 2k) + (2^k - 1)*c
n - 2k = 0
k = n/2
T(n) = 2^(n/2) * T(0) + (2^(n/2) - 1)*c
= 2^(n/2) * (1 + c) - c
T(n) ~ 2^(n/2) Complexidade O(2^(n/2))

Na sua forma iterativa:

```
fibo(int n)
 if(n <= 1){
                return n;
 int fibo = 1;
 int fiboPrev = 1;
 for(int i = 2; i < n; ++i)
                int temp = fibo;
                fibo += fiboPrev;
                fiboPrev = temp;
 return fibo;
```

Somando temos 3*n + 3, o que gera uma complexidade de O(n)

(B) Geração de todas as permutações de um número.

O escolhido foi o Heap, pois ele tenta reduzir o número de computações ao trocar somente dois elementos de cada vez e garante chegar a todas as permutações. Ele começa definindo um contador i = 0 e vai iterar sobre até gerar (n - 1)! permutações que terminam com cada número da sequência, se o valor de n for ímpar trocamos o primeiro termo com o último e se for par trocamos o i-ésimo termo com o último.

O pseudo-código: disponível em https://en.wikipedia.org/wiki/Heap%27s_algorithm

```
procedure generate(n : integer, A : array of any):
   if n = 1 then
        output(A)                          1
   else
      for i := 0; i < n - 1; i += 1 do     n -1 vezes
         generate(n - 1, A)                T(n-1)
         if n is even then
            swap(A[i], A[n-1])             1
         else
            swap(A[0], A[n-1])             1
         end if
      end for
   end if
```

if n = 1 { T(n) = 1 }
if n > 1 { T(n) = n*T(n-1) + 2 } Cuja complexidade é O(n!)

Versão iterativa:

```
procedure generate(n : integer, A : array of any):
   c : array of int

   for i := 0; i < n; i += 1 do     n vezes
      c[i] := 0                     1
   end for
   output(A)
   i := 0;                          1
   while i < n do                   n vezes
      if  c[i] < i then             1
         if i is even then          1
            swap(A[0], A[i])        1
         else
            swap(A[c[i]], A[i])
         end if
         output(A)

         c[i] += 1                  1
         i := 0                     1
      else
         c[i] := 0                  1
         i += 1                     1
      end if
   end while
```

Complexidade de n*1 + n * 4 (levando em consideração if == verdadeiros)
O(n)

UNIVERSIDADE FEDERAL DE RORAIMA
PROFESSOR (A) : Herbert Oliveira Rocha
DISCIPLINA DE ANALISE DE ALGORITMOS.
ALUNO : Ibukun Chife didier Adjitche

[QUESTÃO – 02]

Defina e dê exemplos:

(A) Grafos.

(B) Grafo conexo, acíclico e direcionado.

(C) Adjacência x Vizinhança em grafos.

(D) Grafo planar.

(F) Grafo completo, clique e grafo bipartido.

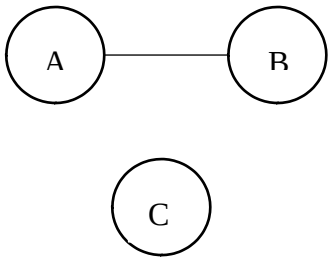
(G) Grafos simples x multigrafo x digrafo.

[RESOLUÇÃO – 02]

(A) Grafos.

Um grafo G, é formado por dois conjuntos de elementos, onde o primeiro conjunto A são os vértices e o segundo conjunto B são as arestas, sendo definido como G = (A, B). Cada aresta liga dois vértices, e cada vértice pode estar ligado a nenhum ou a vários outros vértices.

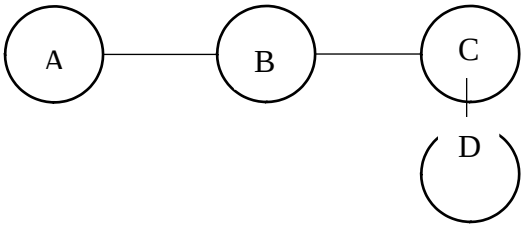
Ex.



(B) Grafo conexo, acíclico e direcionado.

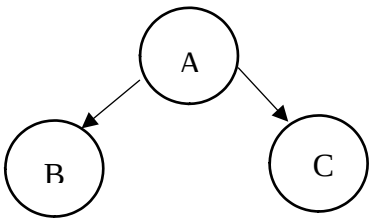
O grafo e conexo quando existe pelo menos um caminho entre qualquer par de vértices.

Ex.



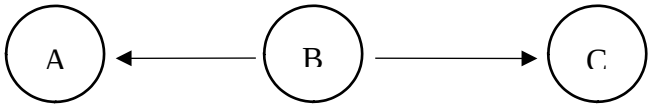
O grafo e acíclico, quando os caminhos entre os vértices não contenha vértices repetidos.

Ex.



O grafo e direcionado, quando as arestas possuem sentido, ou seja, não é permitido percorrer pelas arestas no sentido contrário.

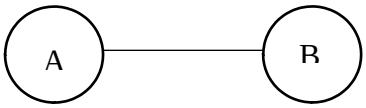
Ex.



(C) Adjacência x Vizinhança em grafos.

Refere-se aos vértices, dois vértices são adjacentes se existe uma aresta ligando os dois.

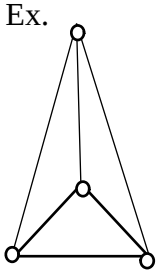
Ex.



(D) Grafo planar.

Um grafo $G(V,A)$ é dito ser planar quando existe alguma forma de se dispor seus vértices em um plano de tal modo que nenhum par de arestas se cruze.

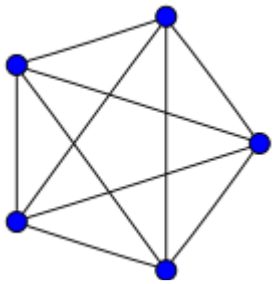
Ex.



(E) Grafo completo, clique e grafo bipartido.

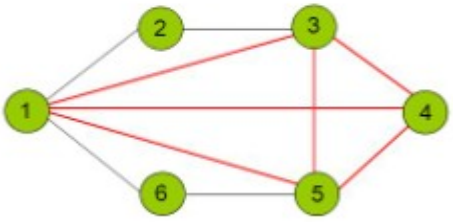
Grafo completo quando há uma aresta entre cada par de seus vértices. Estes grafos são designados por Kn, onde n é a ordem do grafo.

Ex.



Grafo clique é um subgrafo completo

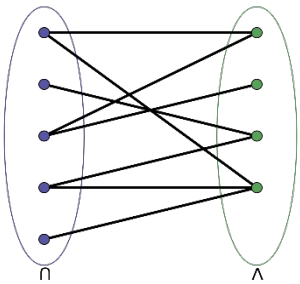
Ex.



UNIVERSIDADE FEDERAL DE RORAIMA
PROFESSOR (A) : Herbert Oliveira Rocha
DISCIPLINA DE ANALISE DE ALGORITMOS.
ALUNO : Ibukun Chife didier Adjitche

Grafo bipartido
Um grafo Kn possui o número máximo possível de arestas para um dados n. Ele é, também regular-(n-1) pois todos os seus vértices tem grau n-1

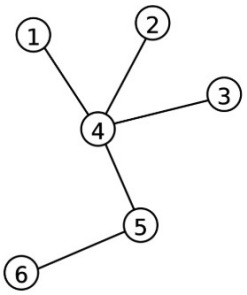
Ex.



(F) Grafos simples x multigrafo x digrafo.

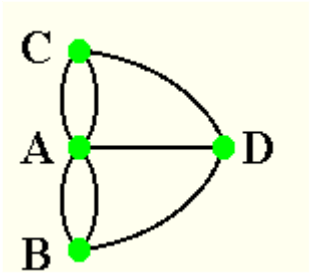
Gráfico simples não passa duas vezes pela mesma aresta (arco).

Ex.



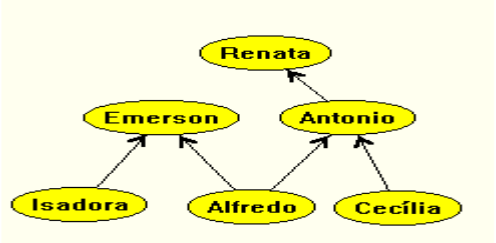
Multigrafo quando existem múltiplas arestas entre pares de vértices de G. No grafo G8, por exemplo, há duas arestas entre os vértices A e C e entre os vértices A e B, caracterizando-o como um multigrafo.

Ex.



Dígrafo pode mais de uma aresta para cada par de vértices e essas arestas possuem direção.

Ex.



UNIVERSIDADE FEDERAL DE RORAIMA
PROFESSOR (A) : Herbert Oliveira Rocha
DISCIPLINA DE ANALISE DE ALGORITMOS.
ALUNO : Ibukun Chife didier Adjitche

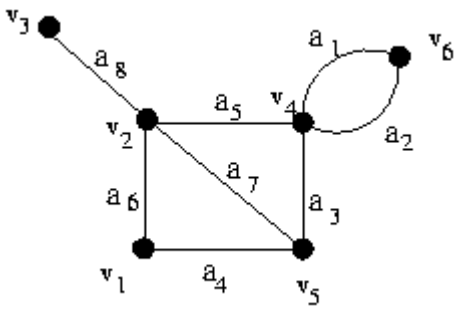
[QUESTÃO – 03]

Defina e apresente exemplos de matriz de incidência, matriz de adjacência e lista de adjacência. Adicionalmente, descreva o impacto (vantagens e desvantagens) da utilização de matriz de adjacência e lista de adjacência.

[RESOLUÇÃO – 03]

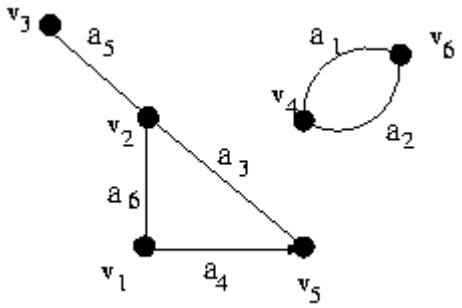
Matriz de incidência

Seja G um grafo de n vértices v_1, v_2... v_n, e m arestas a_1, a_2...a_m, e nenhum laço. A matriz de incidência é uma matriz n x m, onde o valor de cada elemento e_{jk} da matriz é determinado da seguinte maneira:



e_{jk} = 1, se a aresta a_k é incidente ao vértice v_j
= 0 senão

	a1	a2	a3	a4	a5	a6	a7	a8
v1	0	0	0	1	0	1	0	0
v2	0	0	0	0	1	1	1	1
v3	0	0	0	0	0	0	0	1
v4	1	1	1	0	1	0	0	0
v5	0	0	1	1	0	0	1	0
v6	1	1	0	0	0	0	0	0



Eis as propriedades interessantes da matriz de incidência:
1- Como cada aresta é incidente a exatamente dois vértices, cada coluna contém exatamente dois 1.
2- O número de 1 em cada linha é igual ao grau do vértice correspondente.
3- Uma linha que contém somente 0 representa um vértice isolado.
4- Arestas paralelas resultam em duas colunas idênticas.
5- Se um grafo G é desconexo e contém dois componentes g1 e g2, a matriz de incidência A(G) pode ser escrita da seguinte maneira:

Multi-grafo

A representação de um multi-grafo não exige nenhum tratamento especial. Simplemente, a matriz conterá algumas colunas idênticas que correspondem a arestas paralelas.

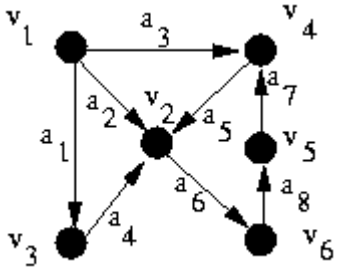
Pseudo-grafo

Para representar um grafo que contém um laço, simplesmente teremos, na coluna que corresponde à aresta, um 1 só que identifica o vértice que contém esse laço. Nessa representação, perdemos a propriedade de ter dois 1 em cada coluna. Uma consequência disso é que será mais difícil detectar erros na construção da matriz (ou na transmissão).

UNIVERSIDADE FEDERAL DE RORAIMA
PROFESSOR (A) : Herbert Oliveira Rocha
DISCIPLINA DE ANALISE DE ALGORITMOS.
ALUNO : Ibukun Chife didier Adjitche

Grafo direcionado

Nesse caso, é preciso distinguir as arestas convergentes das arestas divergentes. Em outras palavras, para cada aresta, temos que especificar de qual vértice ela vem e no qual ela chega. Podemos simplesmente utilizar 1 no primeiro caso e -1 no segundo. Veja a representação do grafo da figura 3:



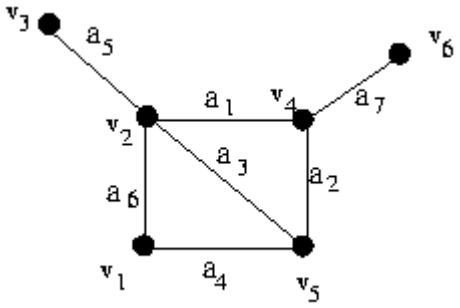
	a1	a2	a3	a4	a5	a6	a7	a8
v1	1	1	1	0	0	0	0	0
v2	0	-1	0	-1	-1	1	0	0
v3	-1	0	0	1	0	0	0	0
v4	0	0	-1	0	1	0	-1	0
v5	0	0	0	0	0	0	1	-1
v6	0	0	0	0	0	-1	0	1

Grafo rotulado em arestas

Quando as arestas tém um valor associado (ou qualquer outro rótulo), uma modificaçao possível é colocar esse valor ao invéz de 1. Mas isso tem a desvantagem de ser redundante, pois o valor seria colocado duas vezes para cada arestas. Para usar menos espaço de memória seria melhor associar em uma tabela separada cada aresta com o seu rótulo.

Matriz de adjacência

Seja G um grafo simples de n vértices v_1, v_2... v_n. A matriz de adjacência é uma matriz n x n, onde o valor de cada elemento e_{jk} da matriz é determinado da seguinte maneira:



e_{jk} = 1, se os vértices v_j e v_k são ligados por uma aresta
= 0 senão

Eis a matriz de adjacência do grafo da figura 4.

	v1	v2	v3	v4	v5	v6
v1	0	1	0	0	1	0
v2	1	0	1	1	1	0
v3	0	1	0	0	0	0
v4	0	1	0	0	1	1
v5	1	1	0	1	0	0
v6	0	0	0	1	0	0

Multi-grafo

Para representar um multi-grafo, é só permitir que um elemento da matriz possa ter um valor superior a 1. Nesse caso, o valor do elemento ejk representa o número de arestas entre v_j e v_k.

Eis as propriedades interessantes da matriz de adjacência:

1- Os valores da diagonal principal da matriz são 0.

2- O grau de um vértice é igual ao número de 1 na linha ou coluna correspondente ao vértice.

3- Permutações de colunas e das linhas correspondentes resultam em uma representação de um grafo isomorfo.

4- Seja um grafo G desconexo que contém dois componentes g1 e g2. A matriz de adjacêncica X(G) pode ser escrita em função das duas matrizes de X(g1) e X(g2):

$$\begin{matrix} X(g1) & 0 \\ 0 & X(g2) \end{matrix}$$

5- Seja qualquer matriz M simétrica, de tamanho n x n, é possível construir um grafo tal que M é a sua matriz de adjacência.

	v1	v2	v3	v4	v5	v6
v1	0	1	1	1	0	0
v2	0	0	0	0	0	1
v3	0	1	0	0	0	0
v4	0	1	0	0	0	0
v5	0	0	0	1	0	0
v6	0	0	0	0	1	0

Peudo-grafo

Não tem dificuldade a representar laços. Simplesmente, um elemento da diagonal principal poderá ter um valor diferente de 0.

Grafo direcionado

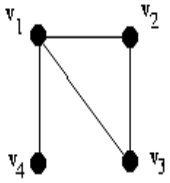
Nesse caso, podemos utilizar uma matriz não necessariamente simétrica. Um elemento e_{jk} da matriz terá o valor 1 se existe uma aresta de v_j até v_k. Eis a representação do grafo da figura 3:

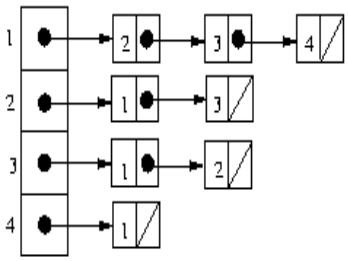
Grafo rotulado em arestas

Para representar um grafo rotulado, podemos simplesmente colocar, ao invéz do valor 1 quando existe uma aresta entre dois vértices, o rótulo associado a essa aresta. Note que essa opção entra em conflito com a representação de multi-grafo. Se for o caso, o valor de cada elemento da matriz deverá ser uma lista.

Estrutura de adjacência

O fato de que as linhas da matriz de adjacência contêm normalmente muitos 0 sugere outras implementações mais compactas. Uma delas é é a estrutura de adjacência. Seja G=(V,E) um grafo. A estrutura de adjacência A de G é um conjunto de n listas A(v), uma para cada vértice v de V. Cada lista A(v) é denominada lista de adjacências de v, e contém todos os vértices que são adjacentes a v.





Eis um exemplo de estrutura de adjacência:

Multi-grafo e Pseudo-grafo

A representação de um multi-grafo ou pseudo-grafo não exige nenhum tratamento especial. A lista de adjacência de um vértice vi conterá dois elementos idênticos no primeiro caso, e vi no segundo caso.

Grafo direcionado

Na definição dada acima, a representação de cada aresta exige dois elementos, um em cada lista de adjacência que corresponde a um vértice ligado pela aresta. Em um grafo direcionado, teria somente um elemento por cada aresta. Poderíamos, por exemplo, dar para cada vértice vi a lista dos vértices ligados por uma aresta que a partir de vi. Poderíamos também, para cada vértice, utilizar duas listas: uma para as arestas divergentes e outra para as arestas convergentes.

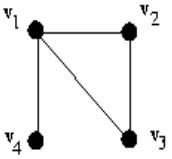
UNIVERSIDADE FEDERAL DE RORAIMA
PROFESSOR (A) : Herbert Oliveira Rocha
DISCIPLINA DE ANALISE DE ALGORITMOS.
ALUNO : Ibukun Chife didier Adjitche

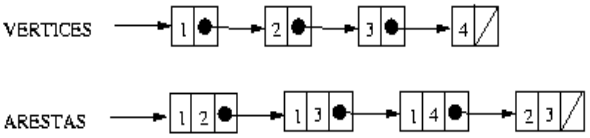
Grafo rotulado em arestas

Nesse caso, simplesmente acrescentamos mais uma informação na estrutura que representa um elemento de lista de adjacência.

Conjuntos

Em certos casos, pode ser vantajoso de representar os grafos de maneira semelhante à definição, isto é, usando dois conjuntos, um para representar os vértices, outro para representar as arestas. Uma maneira de representar um conjunto é o uso de listas. Para representar o conjunto das arestas, os elementos do conjunto serão pares, conforme a definição.
Eis um exemplo:





Multi-grafo

Nesse caso, não podemos representar uma aresta especificando somente os dois vértices que ela liga. É preciso também rotular as arestas.

Pseudo-grafo

A representação de um laço não apresenta dificuldade adicional.

Grafo direcionado

Nesse caso, só precisamos tomar cuidado com a ordem dos elementos que constituem os pares do conjunto de arestas.

Grafo rotulado em arestas

Nesse caso, simplesmente acrescentamos mais uma informação na estrutura que representa uma aresta no conjunto de arestas.

Complexiade na Utilização de Matriz de Adjacência e da lista de Adjacência

complexidade das implementações descritas acima (Supondo *n* o número de vértices e *m* o número de arestas (é colocado em negrito, para cada caso, a representação que geralmente é a melhor):

	Matriz de Adjacência	Lista de Adjacência
Memória	$O(n^2)$	**O(m+n)**
Buscar todos os vizinhos de v_i	O(n)	**O(n)**
Conferir adjacência de v_i e v_j	**O(1)**	O(n)
Visitar todas as arestas	$O(n^2)$	**O(m)***
Calcular grau de um vértice	O(n)	**O(n)**

UNIVERSIDADE FEDERAL DE RORAIMA
PROFESSOR (A) : Herbert Oliveira Rocha
DISCIPLINA DE ANALISE DE ALGORITMOS.
ALUNO : Ibukun Chife didier Adjitche

[QUESTÃO – 04]

Defina, explicando as principais características e exemplifique:

(A) Enumeração explícita x implícita.

(B) Programação Dinâmica.

(C) Algoritmo Guloso.

(D) Backtracking.

[RESOLUÇÃO – 04]

Defina, explicando as principais características e exemplifique:

Enumeração explícita x implícita.

O método de enumeração explicita, faz uma enumeração de todas as possíveis soluções, já em enumeração implícita faz uma enumeração "inteligente", com as melhores soluções para o problema.

Programação Dinâmica.

É um esquema de enumeração de soluções, em utiliza de uma técnica de decomposição minimizar o montante de computação, assim evita que um mesmo subproblema seja resolvido diversas vezes. Resolve o subproblema uma vez e reutiliza a solução toda vez que o mesmo problema aparece.

Algoritmo Guloso.

Sempre fazer a melhor escolha local para ter uma melhor escolha global, geralmente atingindo soluções bem próximas da ótima, porém em alguns casos não encontra soluções tão ótimas. Um exemplo de utilização de algoritmos gulosos é encontrar uma solução do problema da mochila fracionária.

Backtracking.

É um tipo de algoritmo que representa um refinamento da busca por força bruta, em que múltiplas soluções podem ser eliminadas sem serem explicitamente examinadas.

UNIVERSIDADE FEDERAL DE RORAIMA
PROFESSOR (A) : Herbert Oliveira Rocha
DISCIPLINA DE ANALISE DE ALGORITMOS.
ALUNO : Ibukun Chife didier Adjitche

[QUESTÃO – 05]

Implemente uma solução para multiplicação de matrizes utilizando programação dinâmica, visando determinar uma ordem em que as matrizes sejam multiplicadas, de modo a minimizar o número de multiplicações envolvidas.

[RESOLUÇÃO – 05]

```
//Algoritmo de Strassem para multiplicação de Matrizes
//O algoritmo foi testado na plataforma
https://www.jdoodle.com/c-online-compiler na linguagem
C.
//Referenca em :
https://sahebg.github.io/computersceince/matrix-multipli
cation-chain-c-program/

#include <stdio.h>
#include<limits.h>
#define INFY 999999999
long int m[20][20];
int s[20][20];
int p[20],i,j,n;

void print_optimal(int i,int j)
{
if (i == j)
printf(" A%d ",i);
else
  {
    printf("( ");
    print_optimal(i, s[i][j]);
    print_optimal(s[i][j] + 1, j);
    printf(" )");
  }



int MatrixChainOrder(int p[], int i, int j)
{
   if(i == j)
      return 0;
   int k;
   int min = INT_MAX;
   int count;

   for (k = i; k <j; k++)
   {
      count = MatrixChainOrder(p, i, k) +
            MatrixChainOrder(p, k+1, j) +
            p[i-1]*p[k]*p[j];

      if (count < min)
         min = count;
   }

   // Return minimum count
   return min;
}
```

```
void matmultiply(void)
{
long int q;
int k;
for(i=n;i>0;i--)
 {
  for(j=i;j<=n;j++)
   {
    if(i==j)
      m[i][j]=0;
    else
     {
      for(k=i;k<j;k++)
      {
       q=m[i][k]+m[k+1][j]+p[i-1]*p[k]*p[j];
       if(q<m[i][j])
        {
          m[i][j]=q;
          s[i][j]=k;
        }
      }
     }
   }
 }
}


void main()
{
int k;
printf("Enter the no. of elements: ");
scanf("%d",&n);
for(i=1;i<=n;i++)
for(j=i+1;j<=n;j++)
{
 m[i][i]=0;
 m[i][j]=INFY;
 s[i][j]=0;
}
printf("\nEnter the dimensions: \n");
for(k=0;k<=n;k++)
{
 printf("P%d: ",k);
 scanf("%d",&p[k]);
}
matmultiply();
printf("\nCost Matrix M:\n");
for(i=1;i<=n;i++)
 for(j=i;j<=n;j++)
  printf("m[%d][%d]: %ld\n",i,j,m[i][j]);


i=1,j=n;
printf("\nMultiplication Sequence : ");
print_optimal(i,j);
printf("\nMinimum number of multiplications is : %d ",
                    MatrixChainOrder(p, 1, n));

}
```

UNIVERSIDADE FEDERAL DE RORAIMA
PROFESSOR (A) : Herbert Oliveira Rocha
DISCIPLINA DE ANALISE DE ALGORITMOS.
ALUNO : Ibukun Chife didier Adjitche

[QUESTÃO – 06]

Defina e exemplifique:

(A) Problema SAT x Teoria da NP-Completude.

(B) Classes P, NP, NP-Difícil e NP-Completo.

[RESOLUÇÃO – 06]

(A) Problema SAT x Teoria da NP-Completude.

Problemas SAT ou Problemas de satisfabilidade booleana, é o primeiro problema classificado com a complexidade NP-Completo, estar nessa classe de complexidade diz duas coisas sempre, primeiro,que ele está na classe NP e segundo, todos os outros problemas na classe NP poderiam ser reduzidos à esse problema em tempo polinomial, ao ponto que se for possível resolvê-lo, seria possível resolver todos os outros problemas nessa classe em tempo polinomialmente definido.

O problema de Satisfabilidade diz que, dado uma sequência de elementos booleanos (ex.: A ∧ ¬ B) procura-se saber se existem valores possíveis para substituir as variáveis de modo a gerar uma solução verdadeira (TRUE), nesse caso ela é satisfatória. Caso não exista tal combinação, ela retorna falso (FALSE) e é insatisfatória.

Esse problema poderia ser pensando na forma de um grafo se adaptássemos as sentenças para uma forma que exprima implicância, exemplo (a v b) pode escrito como (~a -> b v ~b -> a). Daqui podemos escrever toda uma sentença em forma de grafo e fazer uma pesquisa nesse grafo para achar uma inconsistência, como ~b -> b, que classificaria o problema como insatisfatório e, caso contrário, satisfatório.

(B) Classes P, NP, NP-Difícil e NP-Completo.

A classe P de complexidade abrange problemas cujo as soluções podem ser encontradas em tempo polinomial determinístico. Exemplo, o problema de determinar se um número é primo.
NP (polinomial não determinista) é a classe de problemas cuja solução pode ser verificada a partir de uma entrada e retornar um “sim” ou “não” em um tempo limitado por um polinômio, assim chamados problemas polinomialmente verificáveis.

Exemplo, o problema do isomorfismo de subgrafos: dados dois grafos A e B como input, quer-se determinar se o B é subgrafo de A.

Também dentro de NP existe uma classe chamada NP-Completo cujos elementos atendem a duas condições:

- estar dentro da classe NP, como já dito.
- todo problema em NP é redutível para esse problema em tempo polinomial. De modo que se conseguíssemos achar uma solução em tempo polinomial para esse problema, poderíamos resolver todos os outros problemas em NP em tempo polinomial.

O próprio problema de satisfabilidade, assim como o problema de Coloração de grafos que se propõe a descobrir como colorir os vértices de um grafo baseado em alguma restrição (geralmente que duas cores não podem estar justapostas).

NP-Difícil é a classe dos problemas “tão difíceis quanto NP-completo” por que mesmo não estando na classe NP e mesmo sem nem precisar ser problemas de decisão, estes problemas são reduzíveis a problemas NP. Exemplo, o problema da parada que é um problema de decisão que pergunta se dado um programa ele vai terminar ou rodar para sempre.

UNIVERSIDADE FEDERAL DE RORAIMA
PROFESSOR (A) : Herbert Oliveira Rocha
DISCIPLINA DE ANALISE DE ALGORITMOS.
ALUNO : Ibukun Chife didier Adjitche

[QUESTÃO–7]

Descreva a redução (prove a NP-Completude) do problema do SAT ao Clique. Apresente o pseudo-código do algoritmo NP e mostre graficamente as instâncias e soluções, no processo de redução.

[RESOLUÇÃO–07]