1 Caminho mínimo entre todos os pares de vértices com Floyd-Warshall

Nós temos um grafo e queremos encontrar o menor caminho a partir de u até v para todos os pares (u, v). Note que esta tarefa pode ser resolvida $O(n^3)$, executando o algoritmo de Djikstra $O(n^2)$ para cada um dos n vértices. Contudo, um algoritmo Floyd-Warshall faz o mesmo trabalho em 4 linhas.

```
int graph[128][128], n; // a weighted graph and its size
void floydWarshall() {
  for( int k = 0; k < n; k++ )
    for( int i = 0; i < n; i++ )
      for( int j = 0; j < n; j++ )
        graph[i][j] = min( graph[i][j], graph[i][k] + graph[k][j] );
}
int main {
  // initialize the graph[][] adjacency matrix and
  //n adjacency matrix and n
  // graph[i][i] should be zero for all i.
  // graph[i][j] should be "infinity" if edge (i, j) does not exist
  // otherwise, graph[i][j] is the weight of the edge (i, j)
  floydWarshall();
  // now graph[i][j] is the length of the shortest path from i to j
}
```

Inicialização:

$graph[i][j]$ deve ser inicializado com o peso da aresta (i,j). Para as arestas não existentes, o valor deve ser inicializado com INF. O valor de INF não pode ser maior que $INT\_MAX/2$ para evitar overflow. O algoritmo tem duas entradas: $graph[][]$ e o número de vertices n. $graph[][]$ é modificado para guardar o comprimento do menor caminho entre todos os pares de vértices. Este algoritmo funciona para grafos direcionados e não-direcionados. A ordem dos 3 loops é importante. Note que a ordem i,j,k não funciona.

2 Fecho transitivo de uma relação com Floyd-Warshall

```
bool graph[128][128], n;
int transitividade() {
  for( int k = 0; k < n; k++ )
    for( int i = 0; i < n; i++ )
      for( int j = 0; j < n; j++ )
        graph[i][j] = graph[i][j] || (graph[i][k] && graph[k][j]);
}
```

Depois transitividade(), $graph[][]$ armazena o fecho transitivo da relação inicial, ou seja, $graph[i][j]$ é verdade se existe um caminho a partir de i até j. A função acima pode ser modificada para verificar se uma relação tem a propriedade transitiva.

3 Floyd-Warshall com recuperação de caminho

O Algoritmo de Floyd-Warshall pode ser modificado para facilitar o processo de recuperação do caminho mínimo. $parent[i][j]$ representa o último vértice antes de j no caminho mínimo de i até j. $parent[i][j]$ é setado com i quando temos uma aresta i até j, caso contrário, é setado com -1. O processo de atualização do vetor parent é realizado da seguinte maneira: "Se o caminho de i até j passando por k"é melhor que o atual, então setamos $parent[i][j]$ com o $parent[k][j]$

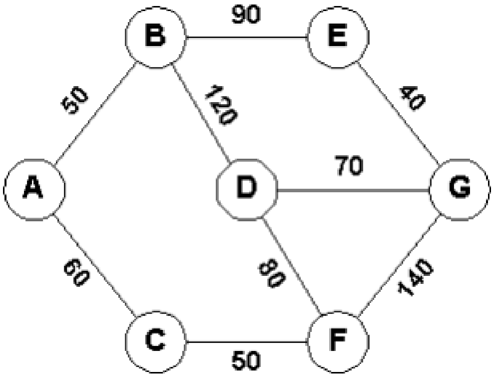
```
int graph[128][128], n; // a weighted graph and its size
int parent[128][128];
void floydWarshall() {
  for( int i = 0; i < n; i++ )
    for( int j = 0; j < n; j++ )
      if ( i == j || graph[i][j] == Inf )
        parent[i][j] = -1;
```

```
      else parent[i][j] = i;

  for( int k = 0; k < n; k++ )
    for( int i = 0; i < n; i++ )
      for( int j = 0; j < n; j++ ) {
        int newD = graph[i][k] + graph[k][j];
        if( newD < graph[i][j] ) {
          graph[i][j] = newD;
          parent[i][j] = parent[k][j];
        }
      }
}
```

4 Problema minmax

Considere um mapa de uma cidade onde as arestas representam ruas e os vértices representam cruzamento. Um inteiro em cada aresta representa o nível de intensidade média do som da rua correspondente. Determine o menor nível de intensidade do som que você deve tolerar para conseguir ir de um cruzamento para outro.



Por exemplo, para ir do cruzamento A até G, podemos seguir os seguintes caminhos: ACFG (maior intensidade 140 decibéis), ABEG(90 decibéis),ABDG(120 decibéis) e ACFDG(80 decibéis). O caminho mais confortável é o caminho com o menor nível de intensidade do som.

```
int intensidade[128][128], n; // a weighted graph and its size
void floydWarshall() {
  for( int k = 0; k < n; k++ )
    for( int i = 0; i < n; i++ )
      for( int j = 0; j < n; j++ )
        intensidade[i][j] = min( intensidade[i][j], max(intensidade[i][k],intensidade[k][j]) );
}
```

Outros exemplos:

- Maximizar a carga transportada por um caminhão quando as estradas ao longo de um caminho podem ter um limite de peso.
- Encontra uma rota em uma rede que atenda um requisito mínimo de largura de banda para algum aplicativo.

5 Caminho mais seguro

Neste problema, precisamos maximizar a probabilidade de sobrevivência entre um caminho.

```
// Safest path variant of Floyd-Warshall example
// input: p is an probability matrix (probability of survival) for n nodes
//     e.g. p[i][j] is the probability of survival moving directly from i to j.
//     the probability from a node to itself e.g. p[i][i] should be initialized to 1
// output: p[i][j] will contain the probability of survival using the safest path from i to j.
for (k=0; k<n; k++)
```

```
for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    p[i][j] = max(p[i][j], p[i][k] * p[k][j]);
```

6 Multiplicação da Matriz de Adjacência

Considere o seguinte problema: Dado um grafo com n vértices, quantos passeios distintos de tamanho k existem começando no vértice 1? Este problema pode ser resolvido com a multiplicação de matrizes.

```
for( int i = 0; i < n; i++ )
  for( int j = 0; j < n; j++ ) {
    C[i][j] = 0;
    for( int k = 0; k < n; k++ )
      C[i][j] += A[i][k] * B[k][j];
  }
```

Se A e B são matrizes de adjacência M então as entradas de C são:

$$C[i][j] = \sum_{k=1}^{n} M[i][k] * M[k][j] \tag{1}$$

Se $M[u][v]$ representa o número de passeios de comprimento 1 a partir do vértice u até v então $C[i][j]$ representa o número de passeios de comprimento 2 de i até j. Observe que são tentados todos os valores para os vértices intermediários. $M^P[i][i]$ representa o número de passeios de comprimento p de i ate j.

Multiplicação de matrizes de adjacência pode ser útil, por exemplo, se precisamos computar o conjunto de vértices alcançados por um vértice s em exatamente p passos. Primeiramente, calculamos M^P e encontramos todos os vértices v tal que $M^P[s][v]$ é diferente de zero. Neste caso, o somatório e a multiplicação podem ser simplificados.

```
for( int i = 0; i < n; i++ )
  for( int j = 0; j < n; j++ ) {
    C[i][j] = 0;
    for( int k = 0; k < n; k++ )
      C[i][j]  = C[i][j] || (A[i][k] && B[k][j]);
  }
```

Exercícios:

1. http://www.spoj.com/problems/ARBITRAG/
2. https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=3&page=show_problem&problem=40
3. http://br.spoj.com/problems/MINIMO/
4. http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2010f1p2_reuniao
5. http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2008f1p2_lanche
6. https://www.urionlinejudge.com.br/judge/pt/problems/view/2308