# Programação Competitiva

## Aula 7 - Dijkstra


Caio Caldeira

Universidade Federal de Minas Gerais


25 de Setembro de 2020

# Problema

- Achar o caminho de menor peso entre dois vértices, x e y, em um grafo;
- Achar o menor caminho partindo de x para todos os vértices em um grafo;
- Achar o menor caminho de todos os vértices para um vértice x arbitrário do grafo;

Para tanto, utilizaremos o algoritmo chamado Dijkstra, a ser demonstrado

# Redução do Problema

Suponha que o peso de todas arestas seja igual

- Ex: Achar a saída do labirinto

# Redução do Problema

Suponha que o peso de todas arestas seja igual

- Ex: Achar a saída do labirinto

Nesse caso, o caminho de menor peso é o caminho com menos arestas, e é possível descobrí-lo utilizando uma Busca em Largura (BFS).

# Algoritmo

F $\longleftarrow$ Fila de tuplas com os vértices a serem visitados e a distância do caminho até eles
U $\longleftarrow$ Conjunto de vertices visitados
D $\longleftarrow$ Vetor com distância dos vertices para x

```
begin
    F ⟵ {x, 0}
    U ⟵ ∅
    while F ≠ ∅ do
        v, tam ⟵ F.pop
        if v ∈ U then
            Continue
        U ⟵ v
        for u ∈ Arestas de v do
            if u ∉ U then
                F ⟵ {u, tam + 1 }
                D[u] ⟵ tam + 1
```

**Algorithm 1:** BFS

# Complexidade

- Vamos visitar cada vértice apenas uma vez
- Pegar o elemento da fila é $\mathcal{O}(1)$
- Nosso For itera para cada vértice no grau de saída dele
- $\sum_{i=1}^{V} Gsi$ é igual ao número de arestas, dividido por dois em um grafo não-direcionado
- Complexidade:$\mathcal{O}(V + E)$

# Prova de Corretude

## Lemma

*Nossa fila é ordenada, e visitamos cada vértice em ordem crescente pelo tamanho do caminho*
*Caso Base: Retiramos o vértice inicial da fila e adicionamos todos os vértices adjacentes com tamanho um*
*Caso K: Suponha que nosso lema seja válido para até o K-ésimo caminho. Como a fila estava ordenada, D[k+1] é menor ou igual à todo elemento da fila e D[k] <= D[k+1]. O maior elemento da fila é no máximo igual à D[k] + 1 e qualquer vértice adjacente a K+1 ainda não visitado será adicionado a fila com valor D[k+1] + 1. Como D[k] <= D[k+1] então D[k] + 1 <= D[k+1] + 1.*

# Prova de Corretude

## Demonstração.

Seja D[i] a distância calculada pelo nosso algoritmo do vértice i até o vértice inicial X, e d[i] o caminho ótimo, então D[i] = d[i]
Caso Base é i = X, que verdade por obviedade. D[X] = 0 = d[X]
Caso K: Seja a hipótese verdadeira para todos os vértices até Y, sendo Y o último vértice a ser calculado. Por nosso lema, para todo D[i] calculado anteriormente D[i] <= D[Y]. Suponha, por contradição, que D[Y] > d[Y]. Seja u o último vértice do caminho entre Y e X tal que D[u] = d[u]. Temos que d[Y] ¿= d[u] + 1.

□

# Expandindo o Problema

E o que acontece se nosso grafo tivesse arestas com valores positivos arbitrários?

# Expandindo o Problema

E o que acontece se nosso grafo tivesse arestas com valores positivos arbitrários?

- Nosso lema se invalida, pois o modo que calculamos o tamanho do caminho altera.

# Expandindo o Problema

E o que acontece se nosso grafo tivesse arestas com valores positivos arbitrários?

- Nosso lema se invalida, pois o modo que calculamos o tamanho do caminho altera.
- Podemos resolver esse problema se ao invés de utilizarmos uma fila utilizassemos uma lista e pegassemos o menor elemento da lista a cada iteração. Isso aumentaria a complexidade de pegar o primeiro elemento da lista para $\mathcal{O}(V)$ aumentando o custo do algortimo para $\mathcal{O}(V^2 + E)$

# Expandindo o Problema

E o que acontece se nosso grafo tivesse arestas com valores positivos arbitrários?

- Nosso lema se invalida, pois o modo que calculamos o tamanho do caminho altera.
- Podemos resolver esse problema se ao invés de utilizarmos uma fila utilizassemos uma lista e pegassemos o menor elemento da lista a cada iteração. Isso aumentaria a complexidade de pegar o primeiro elemento da lista para $\mathcal{O}(V)$ aumentando o custo do algortimo para $\mathcal{O}(V^2 + E)$
- Existe uma estrutura de dados chamada Heap, representada na biblioteca STL pela $priority_queue$ que nos permite pegar o menor valor de um arranjo em $O(\log E)$ e inserir um novo valor na estrutura $O(\log E)$.

# `priority_queue`

- `priority_queue` (fila de prioridade ou Heap) é uma estrutura muito similar à `queue`, porém, aqui os elementos são inseridos de modo que o topo da `priority_queue` e seja sempre maior elemento da estrutura.
- Para nossa utilização, utilizaremos um par como tipo da `priority_queue`, portanto é importante notar que a comparação de pares se dá primeiro entre os elementos first dos dois pares e depois entre os elementos second

# priority_queue

- `priority_queue` (fila de prioridade ou Heap) é uma estrutura muito similar à `queue`, porém, aqui os elementos são inseridos de modo que o topo da `priority_queue` e seja sempre maior elemento da estrutura.
- Para nossa utilização, utilizaremos um par como tipo da `priority_queue`, portanto é importante notar que a comparação de pares se dá primeiro entre os elementos first dos dois pares e depois entre os elementos second

# priority_queue

```cpp
priority_queue<pair<int,int> > pq;

pq.push({5,10}), q.push({10,3}), q.push({5,-1});

while(!pq.empty){
    pair<int,int> p = pq.front();
    cout << p.first << " " << p.second << endl;
    pq.pop();
}
```

- Saída:

  10 3

  5 10

  5 -1

# Adaptação do Algoritmo

- Para adaptarmos nossa solução para abranger essa generalização do problema então, devemos alterar nossa fila para uma fila de prioridade

# Adaptação do Algoritmo

- Para adaptarmos nossa solução para abranger essa generalização do problema então, devemos alterar nossa fila para uma fila de prioridade
- Como a fila de prioridade ordena decrescentemente pelo primeiro valor do par, nosso par é da forma {-Tamanho, Índice do Vértice}

# Adaptação do Algoritmo

- Para adaptarmos nossa solução para abranger essa generalização do problema então, devemos alterar nossa fila para uma fila de prioridade
- Como a fila de prioridade ordena decrescentemente pelo primeiro valor do par, nosso par é da forma {-Tamanho, Índice do Vértice}

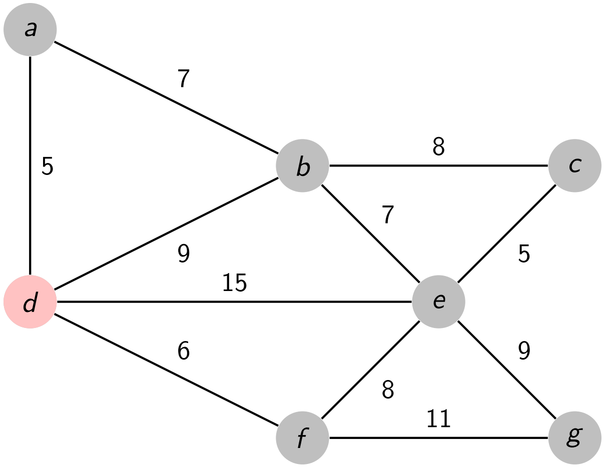
Caso receba erro na representação de par pelas chaves fechadas {}, alterar essa representação para `make_pair`.

# Dijkstra

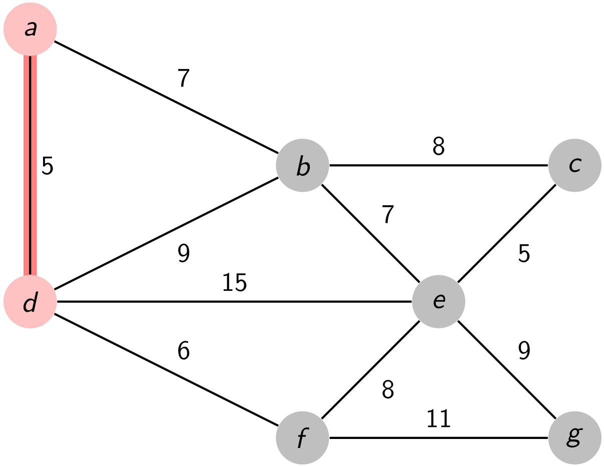
```
vector<pair<int,int> > graph[MAX];
vector<int> dist(MAX, INF);

void dijkstra(int x){
    priority_queue<pair<int,int> pq; pq.push({-0, x});
    while(!pq.empty()){
        int u = pq.front().second, d = -pq.front().first;
        pq.pop();
        if(d > dist[u]) continue;
        dist[u] = d;
        for(pair<int,int> pv: graph[u]){
            int w = d + pv.first;
            if(dist[pv.second] > w)
                pq.push({w, pv.second});
        }
    }
}
```

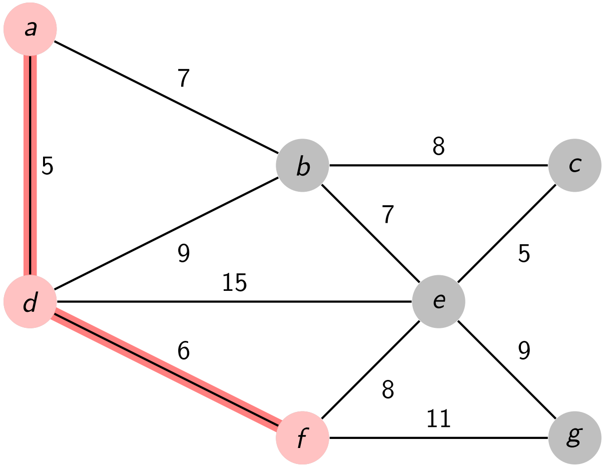
# Executando o Algoritmo



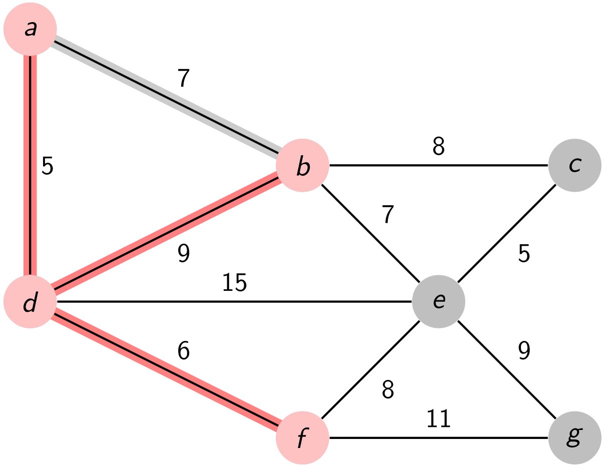
# Executando o Algoritmo



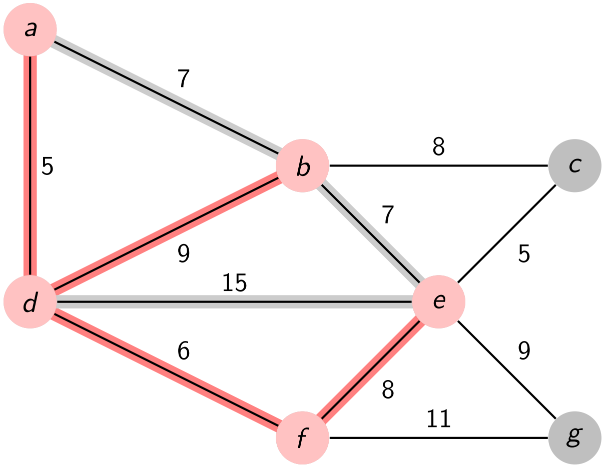
# Executando o Algoritmo



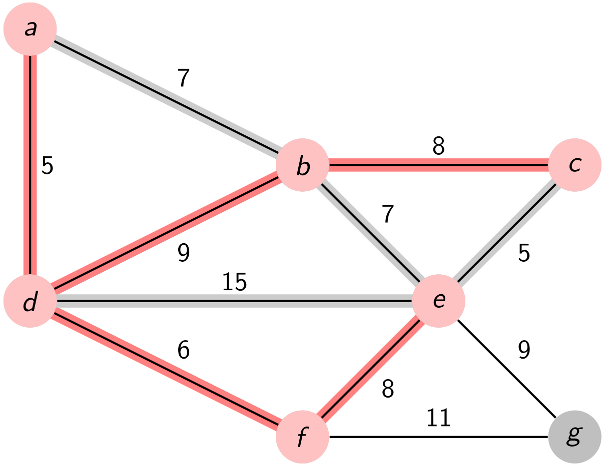
# Executando o Algoritmo



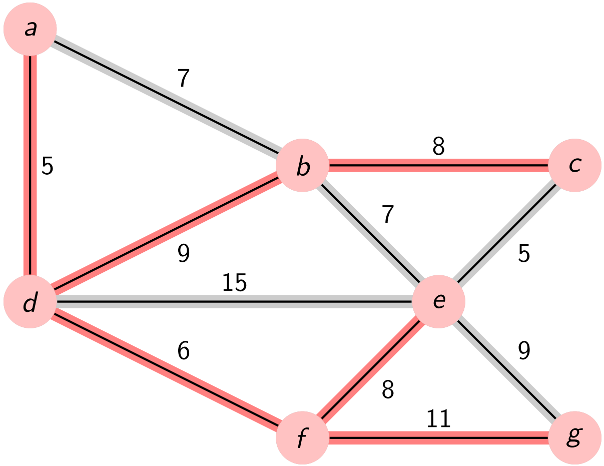
# Executando o Algoritmo



# Executando o Algoritmo



# Executando o Algoritmo



# Motivação: Alocando Ambulâncias

- Existem muitas cidades no reino de Sildávia, mas nem todas possuem hospitais (muitas são bem pequenas, como vilarejos)

# Motivação: Alocando Ambulâncias

- Existem muitas cidades no reino de Sildávia, mas nem todas possuem hospitais (muitas são bem pequenas, como vilarejos)
- Assim, quando uma pessoa necessita de atendimento urgente, o hospital mais próximo envia uma ambulância para socorro (e esta percorre o caminho mínimo para a cidade em que está localizado o paciente)

# Motivação: Alocando Ambulâncias

- Existem muitas cidades no reino de Sildávia, mas nem todas possuem hospitais (muitas são bem pequenas, como vilarejos)
- Assim, quando uma pessoa necessita de atendimento urgente, o hospital mais próximo envia uma ambulância para socorro (e esta percorre o caminho mínimo para a cidade em que está localizado o paciente)
- O governador de Sildávia está muito preocupado com esta situação, e deseja saber qual é o tempo máximo que uma pessoa em Sildávia leva para ser socorrida, e pediu para que você respondesse isso para ele

# Motivação: Alocando Ambulâncias

- Existem muitas cidades no reino de Sildávia, mas nem todas possuem hospitais (muitas são bem pequenas, como vilarejos)
- Assim, quando uma pessoa necessita de atendimento urgente, o hospital mais próximo envia uma ambulância para socorro (e esta percorre o caminho mínimo para a cidade em que está localizado o paciente)
- O governador de Sildávia está muito preocupado com esta situação, e deseja saber qual é o tempo máximo que uma pessoa em Sildávia leva para ser socorrida, e pediu para que você respondesse isso para ele
- O tempo de atendimento tal como o tempo que a ambulância demora para percorrer dentro de uma cidade é desprezado

# Pensando na Solução

- Para solucionar esse problema para apenas uma ambulância deveríamos apenas rodar um Dijkstra partindo da ambulância e o vértice mais distante dessa ambulância teria nossa resposta

# Pensando na Solução

- Para solucionar esse problema para apenas uma ambulância deveríamos apenas rodar um Dijkstra partindo da ambulância e o vértice mais distante dessa ambulância teria nossa resposta
- Podemos utilizar de um conceito ensinado na aula de BFS para solucionar o problema

# Pensando na Solução

- Para solucionar esse problema para apenas uma ambulância deveríamos apenas rodar um Dijkstra partindo da ambulância e o vértice mais distante dessa ambulância teria nossa resposta
- Podemos utilizar de um conceito ensinado na aula de BFS para solucionar o problema
- Uma vantagem do Dijkstra é que não precisamos nos restringir à distância dos vértices até um único vértice. Podemos calcular a distância dos vértices do nosso grafo a todo um conjunto de vértices.

# Pensando na Solução

- Para solucionar esse problema para apenas uma ambulância deveríamos apenas rodar um Dijkstra partindo da ambulância e o vértice mais distante dessa ambulância teria nossa resposta
- Podemos utilizar de um conceito ensinado na aula de BFS para solucionar o problema
- Uma vantagem do Dijkstra é que não precisamos nos restringir à distância dos vértices até um único vértice. Podemos calcular a distância dos vértices do nosso grafo a todo um conjunto de vértices.
- Para isso, basta adicionar todos os vértices do conjunto na nossa fila de prioridade do Dijkstra.

# Resposta

```
vector<pair<int,int> > graph[MAX];
vector<int> dist, amb;
int n, m, q;
void solve();
int dijkstra(int maior = -1);

int main() {_

    while(cin >> n >> m >> q)
        solve();

    return 0;
}
```

# Resposta

```
void solve(){
    for(int i = 0; i < n; i++)
        g[i].clear(), dist[i] = INF, amb[i] = 0;
    for(int i = 0; i < m; i++){
        int a, b, w;
        cin >> a >> b >> w; a--, b--;
        g[a].push_back({b,w});
        g[b].push_back({a,w});
    }
    for(int i = 0; i < q; i++){
        int x; cin >> x; x--;
        amb[x] = 1;
    }
    int maior = dijkstra();
    cout << maior << endl;
}
```

# Resposta

```
int dijkstra(int maior = -1){
    priority_queue<pair<int,int> > pq;
    for(int i = 0; i < n; i++)
        if(amb[i])
            dist[i] = 0, pq.push({0, i});
    while(!pq.empty()){
        int u = pq.top().s, d = -pq.top().f, pq.pop();
        if(d > dist[u]) continue;
        maior = max(maior, dist[u]);
        for(auto pv: g[u])
            if(dist[pv.first] > dist[u] + pv.second){
                dist[pv.first]  = dist[u] + pv.second;
                pq.push({-dist[pv.first], pv.first});
            }
    }
    return maior;
}
```

# Motivação: Didi e a Busca pelo Sorvete

- Após mais uma bem sucedida seletiva interna da UFMG para a Maratona de Programação, Didi encontra-se desesperada por um bom sorvete!

# Motivação: Didi e a Busca pelo Sorvete

- Após mais uma bem sucedida seletiva interna da UFMG para a Maratona de Programação, Didi encontra-se desesperada por um bom sorvete!
- Porém, como é sexta feira, Didi vai aproveitar seu desejo por sorvete e sair com um de seus muitos amigos para por a conversa em dia antes que o semestre na faculdade entre em seu período mais frenético.

# Motivação: Didi e a Busca pelo Sorvete

- Após mais uma bem sucedida seletiva interna da UFMG para a Maratona de Programação, Didi encontra-se desesperada por um bom sorvete!
- Porém, como é sexta feira, Didi vai aproveitar seu desejo por sorvete e sair com um de seus muitos amigos para por a conversa em dia antes que o semestre na faculdade entre em seu período mais frenético.
- Formalmente, Didi tem um grafo conexo com N vértices representando a cidade, no qual ela marcou a UFMG como o vértice 1 e a sorveteria como o vértice N.

# Motivação: Didi e a Busca pelo Sorvete

- Após mais uma bem sucedida seletiva interna da UFMG para a Maratona de Programação, Didi encontra-se desesperada por um bom sorvete!
- Porém, como é sexta feira, Didi vai aproveitar seu desejo por sorvete e sair com um de seus muitos amigos para por a conversa em dia antes que o semestre na faculdade entre em seu período mais frenético.
- Formalmente, Didi tem um grafo conexo com N vértices representando a cidade, no qual ela marcou a UFMG como o vértice 1 e a sorveteria como o vértice N.
- Após uma rápida troca de mensagens, ela marcou também onde cada um de seus X amigos se encontram em seu grafo. Infelizmente, Didi só poderá se encontrar com 1 de seus amigos para tomar sorvete.

# Motivação: Didi e a Busca pelo Sorvete

- Após mais uma bem sucedida seletiva interna da UFMG para a Maratona de Programação, Didi encontra-se desesperada por um bom sorvete!
- Porém, como é sexta feira, Didi vai aproveitar seu desejo por sorvete e sair com um de seus muitos amigos para por a conversa em dia antes que o semestre na faculdade entre em seu período mais frenético.
- Formalmente, Didi tem um grafo conexo com $N$ vértices representando a cidade, no qual ela marcou a UFMG como o vértice 1 e a sorveteria como o vértice $N$.
- Após uma rápida troca de mensagens, ela marcou também onde cada um de seus $X$ amigos se encontram em seu grafo. Infelizmente, Didi só poderá se encontrar com 1 de seus amigos para tomar sorvete.
- Sendo assim, ela gostaria de saber qual o caminho de menor custo da forma $1..x..N$ para algum de seus amigos $x$. Ou seja, ela quer saber qual o valor do menor caminho que vá da UFMG até a casa de algum amigo e, em seguida, para a sorveteria. Note que tal caminho pode repetir vértices!

# Pensando na Solução

- Esse problema pode ser dividido em duas partes

# Pensando na Solução

- Esse problema pode ser dividido em duas partes
- Podemos calcula a distância do vértice inicial para todos os vértices

# Pensando na Solução

- Esse problema pode ser dividido em duas partes
- Podemos calcula a distância do vértice inicial para todos os vértices
- E em seguida calculamos a distância de todos os vértices para a sorveteria

# Pensando na Solução

- Esse problema pode ser dividido em duas partes
- Podemos calcula a distância do vértice inicial para todos os vértices
- E em seguida calculamos a distância de todos os vértices para a sorveteria
- Isso é possível porque o grafo é não-direcional, quais seriam as consequências de o grafo fosse direcional?

# Pensando na Solução

- Esse problema pode ser dividido em duas partes
- Podemos calcula a distância do vértice inicial para todos os vértices
- E em seguida calculamos a distância de todos os vértices para a sorveteria
- Isso é possível porque o grafo é não-direcional, quais seriam as consequências de o grafo fosse direcional?
- Poderíamos resolver o problema invertendo todas as arestas do grafo. Desse modo cada aresta XY passa a significar é possível chegar em X a partir de Y

# Resposta

```cpp
#include <bits/stdc++.h>
#define f first
#define s second
#define pb push_back
#define _ ios::sync_with_stdio(0);cin.tie(0);

using namespace std;

const int INF = 0x3f3f3f3f;
typedef pair<int,int> Edge;
typedef vector<vector<Edge> > Graph;

void add_edge(Graph &g, int from, int to, int weight=1){
    g[from].pb({to, weight});
    g[to].pb({from, weight});
}
```

# Resposta

```
vector<int> dijkstra(Graph &g, int source){
    vector<int> d(g.size(), INF); vector<int> vis(g.size(), 0);

    priority_queue<Edge> pq;
    pq.push({0, source}); d[source] = 0;
    while(!pq.empty()){
        pair<int,int> aux = pq.top(); pq.pop();
        int u = aux.s, w = -aux.f;
        if(d[u] < w or vis[u]) continue; vis[u] = 1;
        for(Edge edg : g[u])
            if(d[edg.f] > w + edg.s){
                d[edg.f] = (w + edg.s);
                pq.push({-d[edg.f], edg.f});
            }
    }
    return d;
}
```

# Resposta

```
int main(){_
    int n, m, x;
    cin >> n >> m >> x;

    Graph grp(n, vector<Edge>());
    Graph rev_grp(n, vector<Edge>());
    vector<int> friends(x);

    for(int i = 0; i < m; i++){
        int a, b, w;
        cin >> a >> b >> w; a--, b--;
        add_edge(grp, a, b, w);
        add_edge(rev_grp, a, b, w);
    }
    for(int &fri: friends)
        cin >> fri;
```

# Resposta

```
    vector<int> to_friends = dijkstra(grp, 0);
    vector<int> to_icecream = dijkstra(rev_grp, n-1);

    pair<int,int> ans = {INF, -1};
    for(int fri: friends){
        int dist = to_friends[fri-1] + to_icecream[fri-1];
        if(ans.f > dist)
            ans = {dist, fri};
    }

    cout << ans.f << endl;
}
```

# Material e Links

- Repositório com Slides e Códigos da aula
- Aula de Dijkstra pela USP
- Vídeo sobre o Dijkstra no Computerphile
- Tutorial de Dijkstra CP-algorithms
- Lista de Exercícios