

Caminhos mínimos

1. Grafos com pesos/custos nas arestas: $G = (V, E)$, $c: E \rightarrow \mathbb{R}_{\geq 0}$
($c(e) \geq 0$ para toda $e \in E$).

Caminhos mínimos

1. Grafos com pesos/custos nas arestas: $G = (V, E)$, $c: E \rightarrow \mathbb{R}_{\geq 0}$ ($c(e) \geq 0$ para toda $e \in E$). **Redes** = *Networks*
2. Queremos: caminhos de peso/custo/comprimento mínimo em G

Variantes do problema

Variantes do problema

- ▶ Dados s e t , queremos um caminho mínimo de s a t em G (problema fonte-sorvedouro).

Variantes do problema

- ▷ Dados s e t , queremos um caminho mínimo de s a t em G (problema fonte-sorvedouro).
- ▷ Dado s queremos um caminho de comprimento mínimo de s a t , para todo $t \in V(G)$ (caminhos mínimos de uma fonte).

Variantes do problema

- ▷ Dados s e t , queremos um caminho mínimo de s a t em G (problema fonte-sorvedouro).
- ▷ Dado s queremos um caminho de comprimento mínimo de s a t , para todo $t \in V(G)$ (caminhos mínimos de uma fonte).
- ▷ Queremos um caminho de comprimento mínimo de s a t , para todo s e para todo $t \in V(G)$ (caminhos mínimos entre todos os pares).

Caminhos mínimos entre todos os vértices

Caminhos mínimos entre todos os vértices

```
void GRAPHspALL(Graph G);  
double GRAPHspDIST(Graph G, int s, int t);  
int GRAPHspPATH(Graph G, int s, int t);
```


Um exemplo de cliente

Um exemplo de cliente

```
void GRAPHdiameter(Graph G)
{ int v, w, vMAX = 0, wMAX = 0;
  double MAX = 0.0;
  GRAPHspALL(G);
  for (v = 0; v < G->V; v++)
    for (w = 0; w < G->V; w++)
      if (GRAPHspPATH(G, v, w) != G->V)
        if (MAX < GRAPHspDIST(G, v, w))
          { vMAX = v; wMAX = w; MAX = GRAPHspDIST(G, v, w); }
  printf("Diameter is %f\n", MAX);
  for (v = vMAX; v != wMAX; v = w)
    { printf("%d-", v); w = GRAPHspPATH(G, v, wMAX); }
  printf("%d\n", w);
}
```

Caminhos mínimos entre todos os vértices, Dijkstra

```
static int st[maxV];
static double wt[maxV];
void GRAPHspALL(Graph G)
{ int v, w; Graph R = GRAPHreverse(G);
  G->dist = MATRIXdouble(G->V, G->V, maxWT);
  G->path = MATRIXint(G->V, G->V, G->V);
  for (v = 0; v < G->V; v++)
  {
    GRAPHpfs(R, v, st, wt);
    for (w = 0; w < G->V; w++)
      G->dist[w][v] = wt[w];
    for (w = 0; w < G->V; w++)
      if (st[w] != -1) G->path[w][v] = st[w];
  }
}
```

Caminhos mínimos entre todos os vértices, Dijkstra (cont.)

```
double GRAPHspDIST(Graph G, int s, int t)
    { return G->dist[s][t]; }
int GRAPHspPATH(Graph G, int s, int t)
    { return G->path[s][t]; }
```

Caminhos mínimos entre todos os vértices, Dijkstra (cont.)

Propriedade 1 (Propriedade 21.7). *Com o algoritmo de Dijkstra, podemos determinar caminhos mínimos entre todos os pares de vértices de um grafo em tempo $O(nm \log n)$.*

Caminhos mínimos entre todos os vértices, Dijkstra (cont.)

Propriedade 1 (Propriedade 21.7). *Com o algoritmo de Dijkstra, podemos determinar caminhos mínimos entre todos os pares de vértices de um grafo em tempo $O(nm \log n)$.*

- ▶ Bom para grafos esparsos. Se m é proporcional a n^2 , podemos usar a técnica de Warshall para remover o fator logarítmico.

Programa 19.3 (Warshall: fecho transitivo, recordação)

```
void GRAPHtc(Graph G)
{ int i, s, t;
  G->tc = MATRIXint(G->V, G->V, 0);
  for (s = 0; s < G->V; s++)
    for (t = 0; t < G->V; t++)
      G->tc[s][t] = G->adj[s][t];
  for (s = 0; s < G->V; s++) G->tc[s][s] = 1;
  for (i = 0; i < G->V; i++)
    for (s = 0; s < G->V; s++)
      if (G->tc[s][i] == 1)
        for (t = 0; t < G->V; t++)
          if (G->tc[i][t] == 1) G->tc[s][t] = 1;
}

int GRAPHreach(Graph G, int s, int t)
{ return G->tc[s][t]; }
```

Caminhos mínimos entre todos os vértices, Floyd

```
void GRAPHspALL(Graph G)
{ int i, s, t;
  double **d = MATRIXdouble(G->V, G->V, maxWT);
  int **p = MATRIXint(G->V, G->V, G->V);
  for (s = 0; s < G->V; s++)
    for (t = 0; t < G->V; t++)
      if ((d[s][t] = G->adj[s][t]) < maxWT) p[s][t] = t;
  for (i = 0; i < G->V; i++)
    for (s = 0; s < G->V; s++)
      if (d[s][i] < maxWT)
        for (t = 0; t < G->V; t++)
          if (d[s][t] > d[s][i]+d[i][t])
            { p[s][t] = p[s][i]; d[s][t] = d[s][i]+d[i][t]; }
  G->dist = d; G->path = p;
}
```


Correção e complexidade de Floyd

Correção e complexidade de Floyd

Propriedade 2 (Propriedade 21.8). *Com o algoritmo de Floyd, podemos determinar caminhos mínimos entre todos os vértices de um grafo em tempo proporcional a n^3 .*

Demonstração. Complexidade: por inspeção. Correção: indução, como em Warshall. □

Correção e complexidade de Floyd

Propriedade 2 (Propriedade 21.8). *Com o algoritmo de Floyd, podemos determinar caminhos mínimos entre todos os vértices de um grafo em tempo proporcional a n^3 .*

Demonstração. Complexidade: por inspeção. Correção: indução, como em Warshall. □

▷ Comparado com Dijkstra, mais eficiente no caso em que m é proporcional a n^2 (removemos o fator logarítmico).

DAGs com pesos nos arcos/Redes acíclicas

DAGs com pesos nos arcos/Redes acíclicas

- ▶ Caminhos mínimos de uma fonte em tempo linear: $O(n + m)$

DAGs com pesos nos arcos/Redes acíclicas

- ▶ Caminhos mínimos de uma fonte em tempo linear: $O(n + m)$
- ▶ Caminhos mínimos entre todos os pares de vértices em tempo $O(nm)$
(custos arbitrários, isto é, custos negativos permitidos)

DAGs com pesos nos arcos/Redes acíclicas

- ▶ Caminhos mínimos de uma fonte em tempo linear: $O(n + m)$
- ▶ Caminhos mínimos entre todos os pares de vértices em tempo $O(nm)$
(custos arbitrários, isto é, custos negativos permitidos)
- ▶ Caminhos mais longos em tempo polinomial

Programa 21.6, Caminhos mais longos

Programa 21.6, Caminhos mais longos

```
static int ts[maxV];
void GRAPHlpt(Graph G, int s, int st[], double wt[])
{ int i, v, w; link t;
  GRAPHts(G, ts);
  for (v = ts[i = 0]; i < G->V; v = ts[i++])
    for (t = G->adj[v]; t != NULL; t = t->next)
      if (wt[w = t->v] < wt[v] + t->wt)
        { st[w] = v; wt[w] = wt[v] + t->wt; }
}
```

Programa 21.7, Caminhos mínimos entre todos os vértices

```
void SPdfsR(Graph G, int s)
{ link u; int i, t; double wt;
  int **p = G->path; double **d = G->dist;
  for (u = G->adj[s]; u != NULL; u = u->next)
  {
    t = u->v; wt = u->wt;
    if (d[s][t] > wt)
      { d[s][t] = wt; p[s][t] = t; }
    if (d[t][t] == maxWT) SPdfsR(G, t);
    for (i = 0; i < G->V; i++)
      if (d[t][i] < maxWT)
        if (d[s][i] > wt+d[t][i])
          { d[s][i] = wt+d[t][i]; p[s][i] = t; }
  }
}
```

Programa 21.7, Caminhos mínimos entre todos os vértices (cont.)

```
void GRAPHspALL(Graph G)
{ int v;
  G->dist = MATRIXdouble(G->V, G->V, maxWT);
  G->path = MATRIXint(G->V, G->V, G->V);
  for (v = 0; v < G->V; v++)
    if (G->dist[v][v] == maxWT) SPdfsR(G, v);
}
```