PAII-11: caminos mas cortos (2)

Dr. J.B. Hayet

CENTRO DE INVESTIGACIÓN EN MATEMATICAS

Marzo 2008

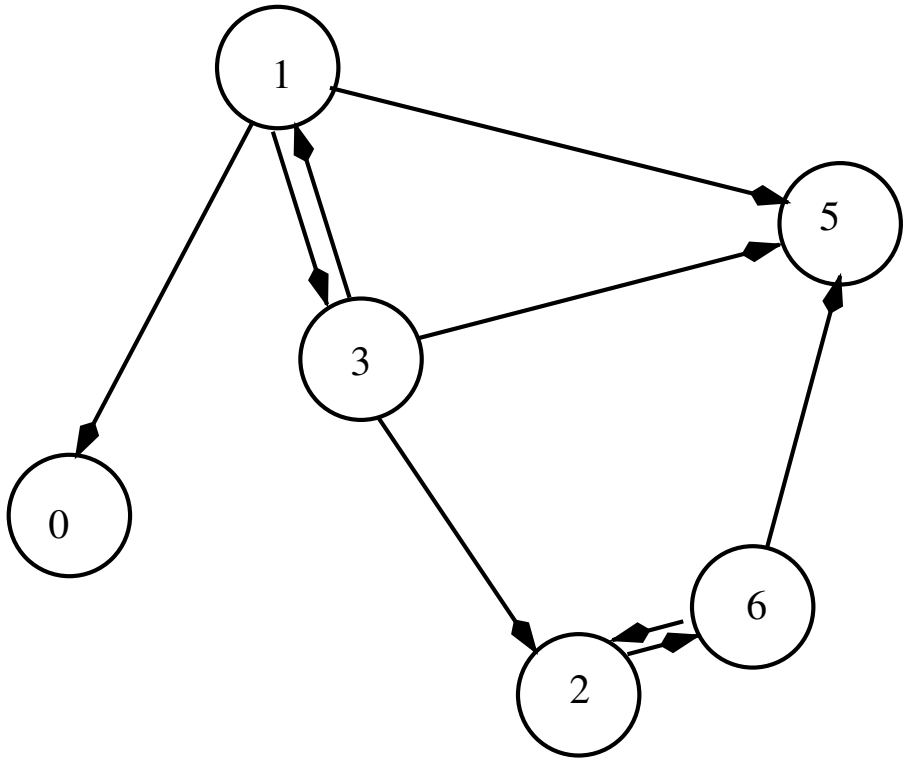
Outline

Outline

Grafos euclidianos

Grafos ponderados en que:

- Vertices son puntos del plano euclidiano $\mathbb{R}^2$
- Aristas son ponderadas por las distancias euclidianas entre los vértices
- Dirigidos eventualmente



Grafos euclidianos

Sobre los caminos más cortos:

- Ponderaciones positivas
- Occure algo particular: para cada par de vértices (v, w) conectados, siempre tendremos que no hay camino más corto de v a w que el arco $v - w$:

$$d[v][w] \leq d[v][x] + d[x][w]$$

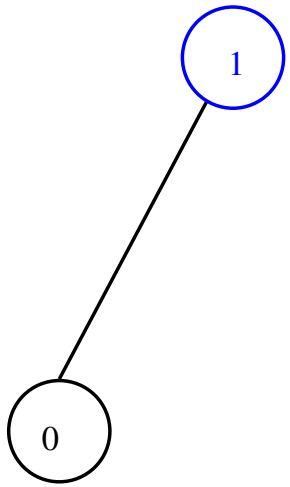
- Además, podemos usar cuotas inferiores sobre el tamaño de los caminos de esta manera:

$$d[v][w] \geq \delta(v, w)$$

Grafos euclidianos

Remarcar que para un grafo ponderado dado, podemos definir dos problemas:

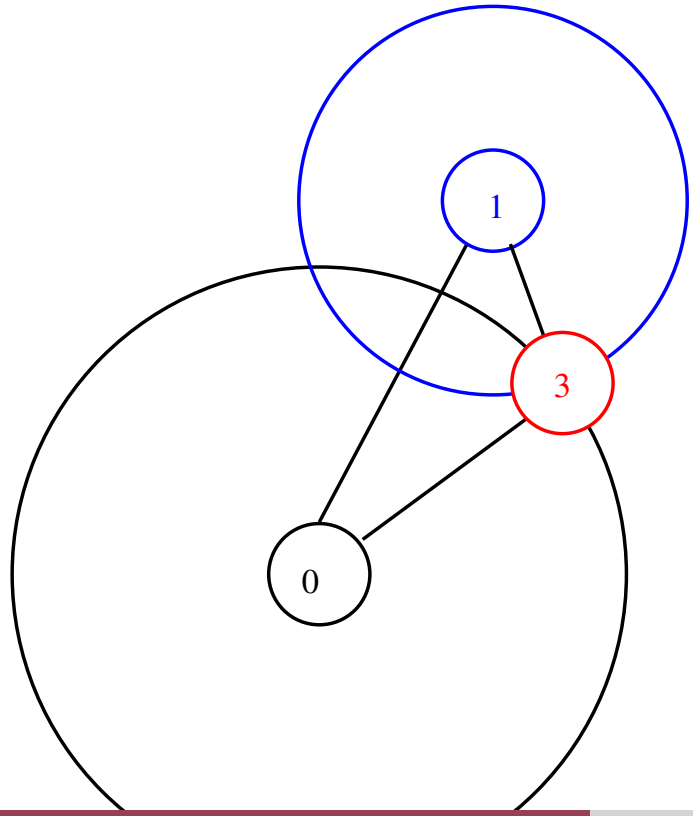
- si corresponde o no a un grafo euclidiano: por ejemplo, las dos propiedades precedentes son condiciones suficientes?
- si sí, cuantas representaciones tendría en el plano euclidiano?



Grafos euclidianos

Remarcar que para un grafo ponderado dado, podemos definir dos problemas:

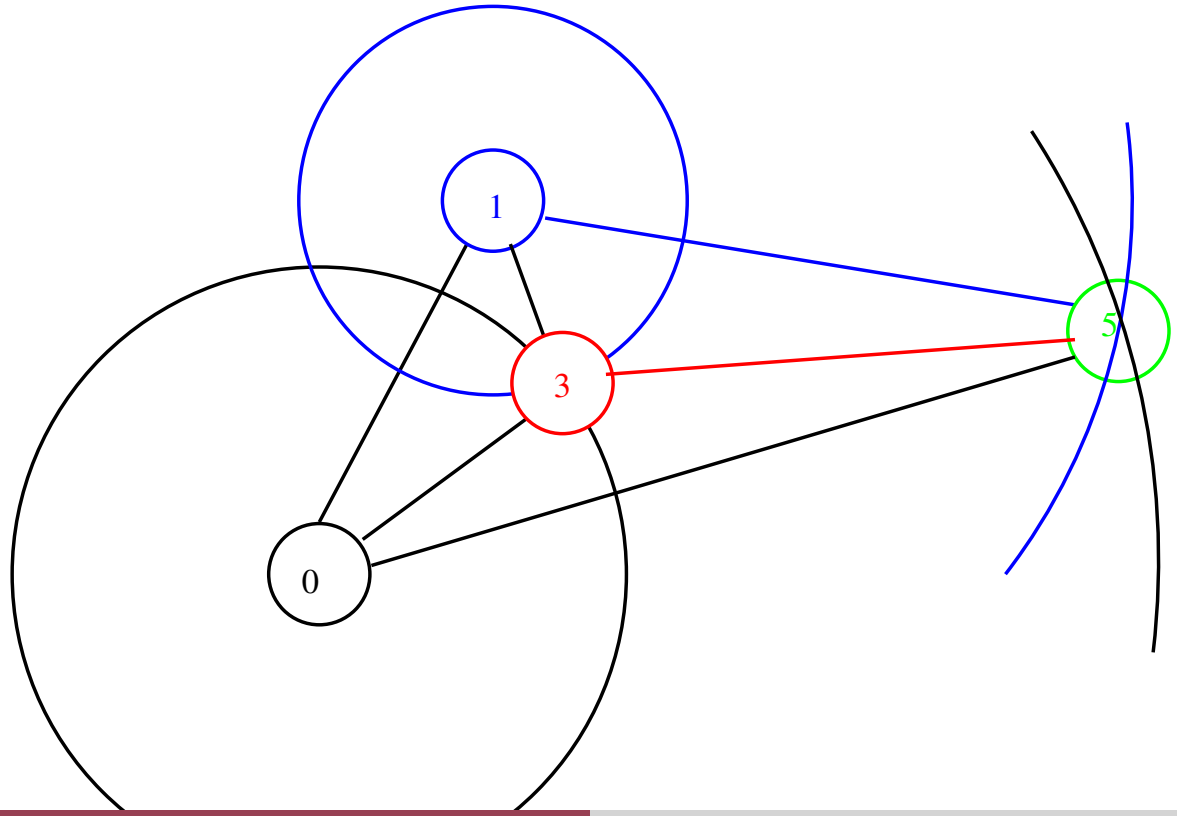
- si corresponde o no a un grafo euclidiano: por ejemplo, las dos propiedades precedentes son condiciones suficientes?
- si sí, cuantas representaciones tendría en el plano euclidiano?



Grafos euclidianos

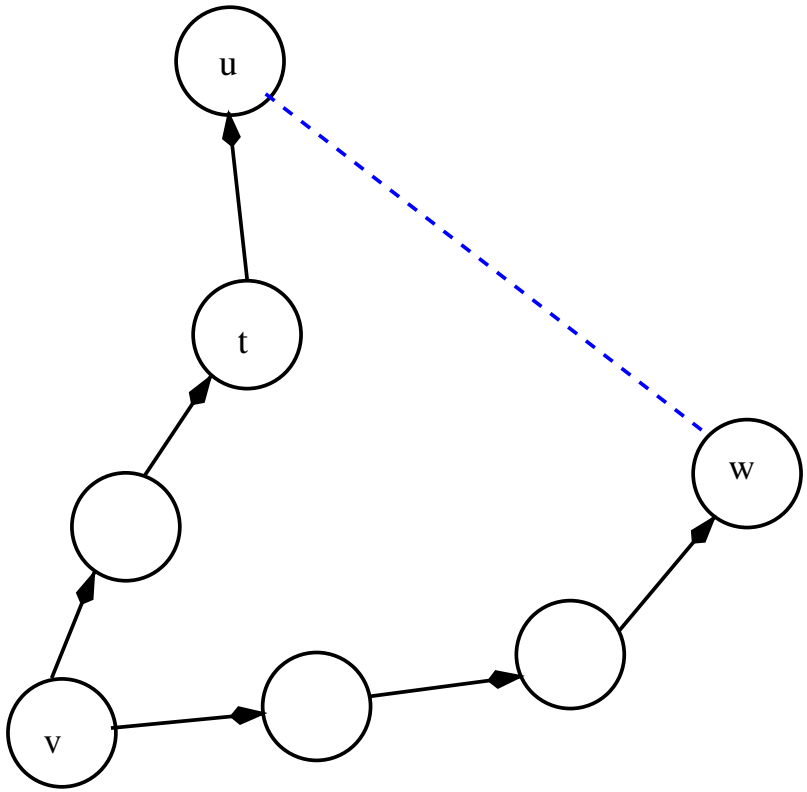
Remarcar que para un grafo ponderado dado, podemos definir dos problemas:

- si corresponde o no a un grafo euclidiano: por ejemplo, las dos propiedades precedentes son condiciones suficientes?
- si sí, cuantas representaciones tendría en el plano euclidiano?



Grafos euclidianos

Consecuencia para correr algoritmos de tipo Dijsktra desde una fuente hasta un vértice final:



Hay unos arcos que puedo descartar automáticamente! Tengo una cuota sobre el mejor camino que voy a poder alcanzar tomando u como próximo: $wt[t] + e- > wt() + \delta(u, w)$

Grafos euclidianos

En un PFS, podemos integrar esta información para seleccionar los arcos siguientes del árbol, tomando como prioridad:

$$P = cmc[t] + e- > wt() + \delta(u, w)$$

- O sea reemplazar el camino más corto hacia los vecinos por el camino con una cuota inferior mínima de la distancia hasta la meta w. Es interesante ya que no necesitamos forzosamente calcular todos los caminos mas cortos sino el entre v y w
- Anticipo un poco sobre los caminos que me van a llegar mas probablemente hasta w

Grafos euclidianos

Implementación de la heurística euclidiana:

- mantener en $wt[t]$ el cuota inferior del tamaño del camino mas corto de v a w pasando por t, que es el tamaño del camino mas corto $+$ la distancia $\delta(t, w)$
- de manera equivalente, $wt[t] - \delta(t, w)$ es el tamaño del camino mas corto hasta t (hasta ahora)
- Inicializar el primer nodo $wt[v]$ a $\delta(v, w)$
- Prioridad:

```
double P = wt[t] + e->wt() +
                   dist(u,w) - dist(t,w);
```

Dijsktra en grafos euclidianos

```
template <class Graph, class Edge> class eucSP{
  const Graph &G;
  vector<double> wt;
  vector<Edge *> sp;
public:
  eucSP(const Graph &G, int v,int w) : G(G),
    sp(G.V()), wt(G.V(), G.V()){
    PQi<double> pQ(G.V(), wt);
    for (int s = 0; s < G.V(); s++) pQ.insert(s);
    wt[v] = dist(v,w); pQ.lower(v);
    while (sp[w]==NULL) {
      int t = pQ.getmin();
      if (t != v && sp[t] == NULL) return;
      typename Graph::adjIterator A(G, v);
```

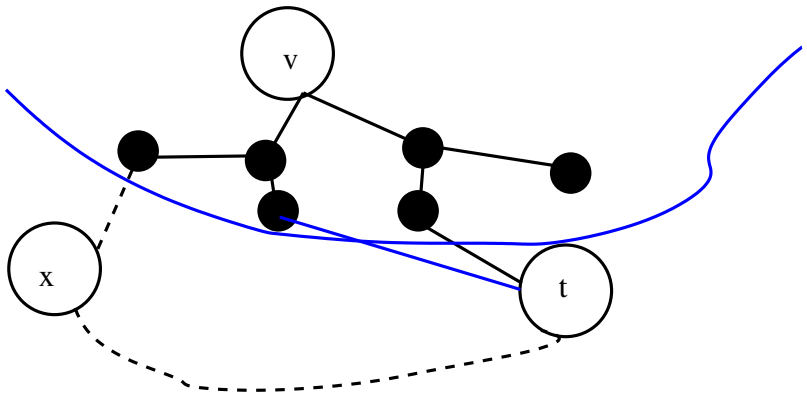
Dijsktra en grafos euclidianos

```
        for (Edge* e = A.beg(); !A.end(); e = A.nxt(
            int u = e->w();
            double P = wt[t] + e->wt() +
            dist(u,w) - dist(t,w);
            if (P < wt[u]) {
             wt[u] = ; pQ.lower(u); sp[u] = e; }
          }
      }
    }
    Edge *pathR(int v) const { return spt[v]; }
    double dist(int v) const { return wt[v]; }
};
```

Grafos euclidianos

Prueba (PFS con heurística euclidiana): visitamos los vértices siguiendo caminos mas cortos: si el camino mas corto es constituido de vértices del árbol, en el momento de elegir un vértice t, éste dará el mínimo de prioridades ya que a todos los que llegan a t les he añadido la misma cantidad; al llegar en t, si el camino mas corto pasa por vértices afuera del árbol (x el primero), este camino tiene como cuota inferior el tamaño del camino de v a x plus $\delta(x,t)$ y es superior al tamaño del camino $v-t$. Llegaríamos a una contradicción:

$$c_{vx} + \delta(x,w) \leq c_{vx} + \delta(x,t) + \delta(t,w) \leq c_{vt} + \delta(t,w)$$



Grafos euclidianos

- Remarcar que se podría resolver de manera equivalente cambiando las ponderaciones de los arcos $e = (t - u)$ por:

$$e->wt() = e->wt() + dist(u, w) - dist(t, w);$$

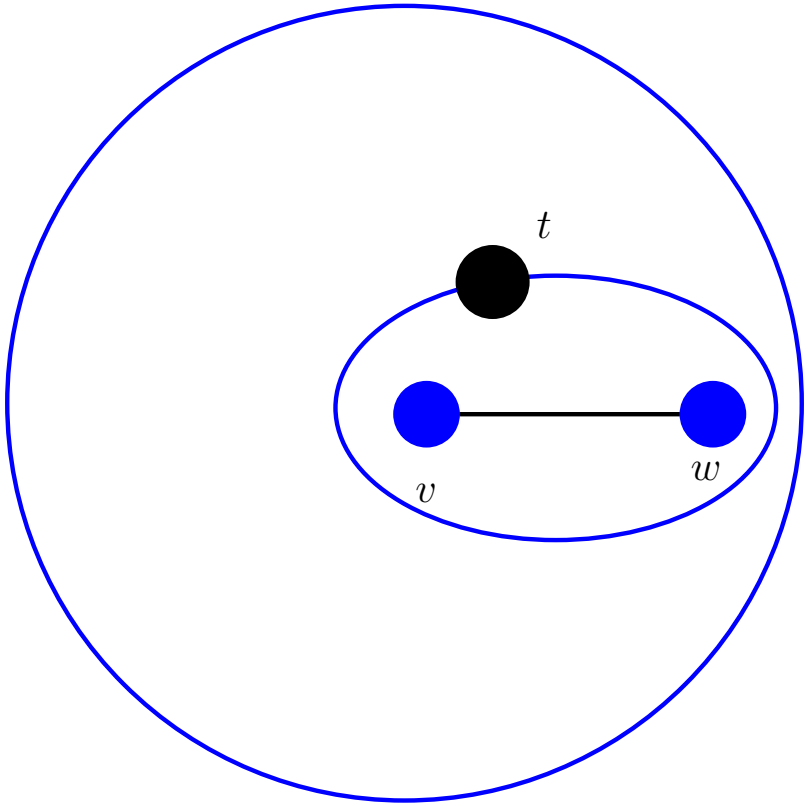
  y usando un Dijsktra normal!
- Es una heurística: no hay ninguna garantía de que se mejore el resultado comparado con un Dijsktra clásico
- Dijsktra: vértices cuyos caminos mas cortos desde v son mas cortos que el hasta w
- Heurística euclidiana: vértices cuyos caminos mas cortos desde v plus la distancia euclidiana a w son mas cortos que el camino $v - w$

Grafos euclidianos

- En un grafo euclidiano debería de haber mucho menos de vértices del segundo tipo!
- La búsqueda esta orientada hasta w
- Se considera que si con Dijsktra el árbol tiene $|V|$ vértices para un grafo euclidiano *random*, el árbol construido con la heurística contendrá solo $\sqrt{V}$ vértices

Grafos euclidianos

Dar direccionalidad:



Es mas generalmente el principio de A^*, que usa cualquier minorante posible al lugar de la simple distancia euclidiana entre el vértice y el punto de destinación; si se elige el mejor minorante disponible, los resultado son óptimos

Outline

Reducción

Un problema A se puede reducir a un problema B si con un algoritmo que resuelve B podemos diseñar un algoritmo que resuelve A, tal que en el peor caso, la complejidad en tiempo no será mas que un factor constante por la complejidad del algoritmo sobre B en el peor caso.

Tres cosas que mostrar para probar que A se reduce a B:

- transformar A en una instancia B
- resolver la instancia B
- transformar la solución de B en solución de A

Reducción

Propiedad: para grafos ponderados con ponderaciones no-negativas, el problema de cerradura transitiva se reduce al problema de calculo de caminos cortos para todos los pares de vértices

Construir un grafo ponderado con ponderaciones de valor dado para cada arco del grafo no ponderado, y ponderaciones con valor-marcador muy grande para las no-aristas; resolver con Floyd; sacar aristas $v - w$ cuando el camino mas corto entre v y w en el grafo intermediario es inferior al valor-marcador. No realmente sorprendiente :)

Reducción

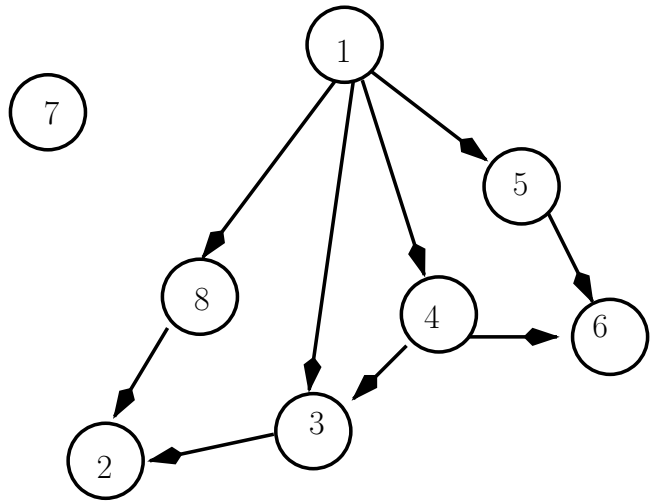
Propiedad: para grafos ponderados sin restricción sobre las ponderaciones (de signo, en particular), los problemas de buscar caminos mas largos y caminos mas cortos son equivalentes

Negar las ponderaciones...

Pero no es cierto si tenemos restricciones sobre los pesos!

Reducción

Problema de job scheduling: dado un conjunto de tareas que hacer, con el tiempo que toma para hacer cada una, y un orden de precedencia, determinar la secuencia de tareas que haga cada uno de esas en un tiempo mínimo



1 0.2
2 0.1
3 0.5
4 0.7
5 0.3
6 0.7
7 0.2
8 0.5

Le puedes añadir mas complejidad (recursos, deadlines. . .)

Reducción

Problema de restricciones de diferencias: asignar n valores no-negativas t_n que minimiza el valor de t_n mientras satisfaciendo un conjunto de restricciones de desigualdad sobre las diferencias, de tipo $t_j - t_i \geq c_{ij}$ donde c_{ij} son constantes

Eso es una versión mas simple de un problema de programación lineal (minimizar una combinación lineal con restricciones en forma de desigualdades sobre otras combinaciones lineales)

Reducción

Propiedad: El problema de job scheduling se reduce al problema de restricciones de diferencias

Definir un nuevo job t_{n+1} que necesitara todos los otros estar hechos antes; el problema de job scheduling sí se escribe entonces como la minimizacion de t_{n+1} el tiempo en que se empieza el nuevo job, con las restricciones de precedencia expresadas como $t_j - t_i \geq d_i$

Equivalencia?

Reducción

Propiedad: El problema de restricciones de diferencias con constantes positivas es equivalente al problema de búsqueda de camino mas largo con fuente única en un grafo dirigido aciclico ponderado

Construir un grafo con cada variable t_i como vértice, y poner las constantes como ponderaciones de los arcos $i - j$; crear un vértice adicional t_0 conectado con peso 0 hasta todos los otros vértices; si hay ciclo de todos modos el sistema no tiene solución; sino, a cada vértice le puedo asociar un camino mas largo hasta el vértice 0 y si asocio su valor a cada vértice, obtengo una solución al problema de diferencias.

Aqui sí hay equivalencia

Reducción

```
typedef WeightedEdge EDGE;
typedef DenseGRAPH<EDGE> GRAPH;
int main(int argc, char *argv[]) {
    int i, s, t, N = atoi(argv[1]);
    double duration[N];
    GRAPH G(N, true);
    for (int i = 0; i < N; i++) cin >> duration[i]
    while (cin >> s >> t)
      G.insert(new EDGE(s, t, duration[s]));
    LPTdag<GRAPH, EDGE> lpt(G);
    for (i = 0; i < N; i++)
      cout << i << " " << lpt.dist(i) << endl;
  }
```

Reducción

- Entonces si no tengo ciclos, puedo usar un algoritmo de búsqueda de caminos mas largos para resolver el scheduling
- Necesito mostrar que no hay ciclos para poder declarar factible el problema de *scheduling*
- Soluciones al *job scheduling* no implican nada para el problema de restricciones sobre diferencias con constantes positivas, y aun menos para el problema general de restricciones sobre diferencias

Reducción

Problema de job scheduling con deadlines: es una version mas especifica del problema de job scheduling en que ademas se añade un conjunto de restricciones sobre tiempos máximos (relativamente a otra tarea) en qué se tienen que empezar tareas

$$t_i \leq t_j + d_{ij}$$

Claramente, las restricciones pueden no estar compatibles y el problema se hace mas "complicado"

Reducción

Propiedad: El problema de *job scheduling* con *deadlines* se reduce al problema de búsqueda de caminos mas cortos mas general (i.e. con pesos negativos autorizados)

Las nuevas restricciones se pueden escribir $t_j - t_i \geq -d_{ij}$, con $d_{ij} \geq 0$; formar un grafo de arcos $i - j$ ponderados con las constantes de las precedencias o de las deadlines; negar los pesos; los caminos mas cortos satisfacen las restricciones

Reducción

- Mala noticia: el problema general de determinación de caminos mas cortos es NP-dificil; eso no implica nada especial sobre el problema de scheduling.
- Pero el problema de job-scheduling con *deadlines* no se podría reducir a una instancia mas especifica del problema de caminos mas cortos?

Outline

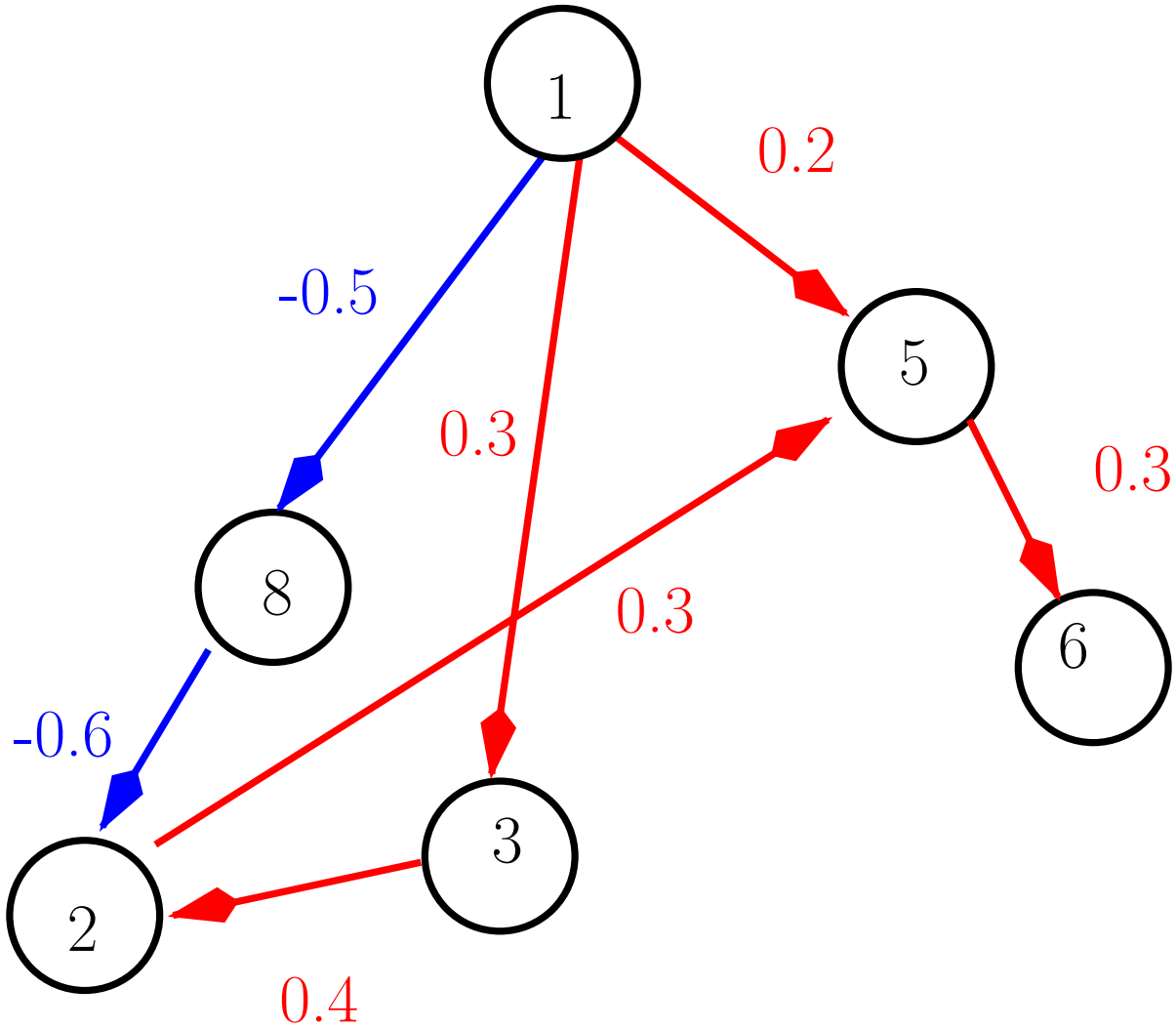
Unas observaciones de buen sentido

- con pesos positivos, buscar caminos mas cortos correspondía a buscar atajos para crear caminos mas chicos
- con pesos negativos, será el contrario, vamos a buscar hacer detours!
- Una idea ingenua: buscar el valor mínimo entre los pesos negativos, y añadir su valor absoluta a todos. Problema?

Unas observaciones de buen sentido

- con pesos positivos, buscar caminos mas cortos correspondía a buscar atajos para crear caminos mas chicos
- con pesos negativos, será el contrario, vamos a buscar hacer detours!
- Una idea ingenua: buscar el valor mínimo entre los pesos negativos, y añadir su valor absoluta a todos. Problema?
- No funciona porque el efecto de esta transformación no se siente de la misma manera según los caminos: el cambio se haría sobre cada arco individualmente, y penalizaría los caminos mas largos en término de número de vertices!

Unas observaciones de buen sentido

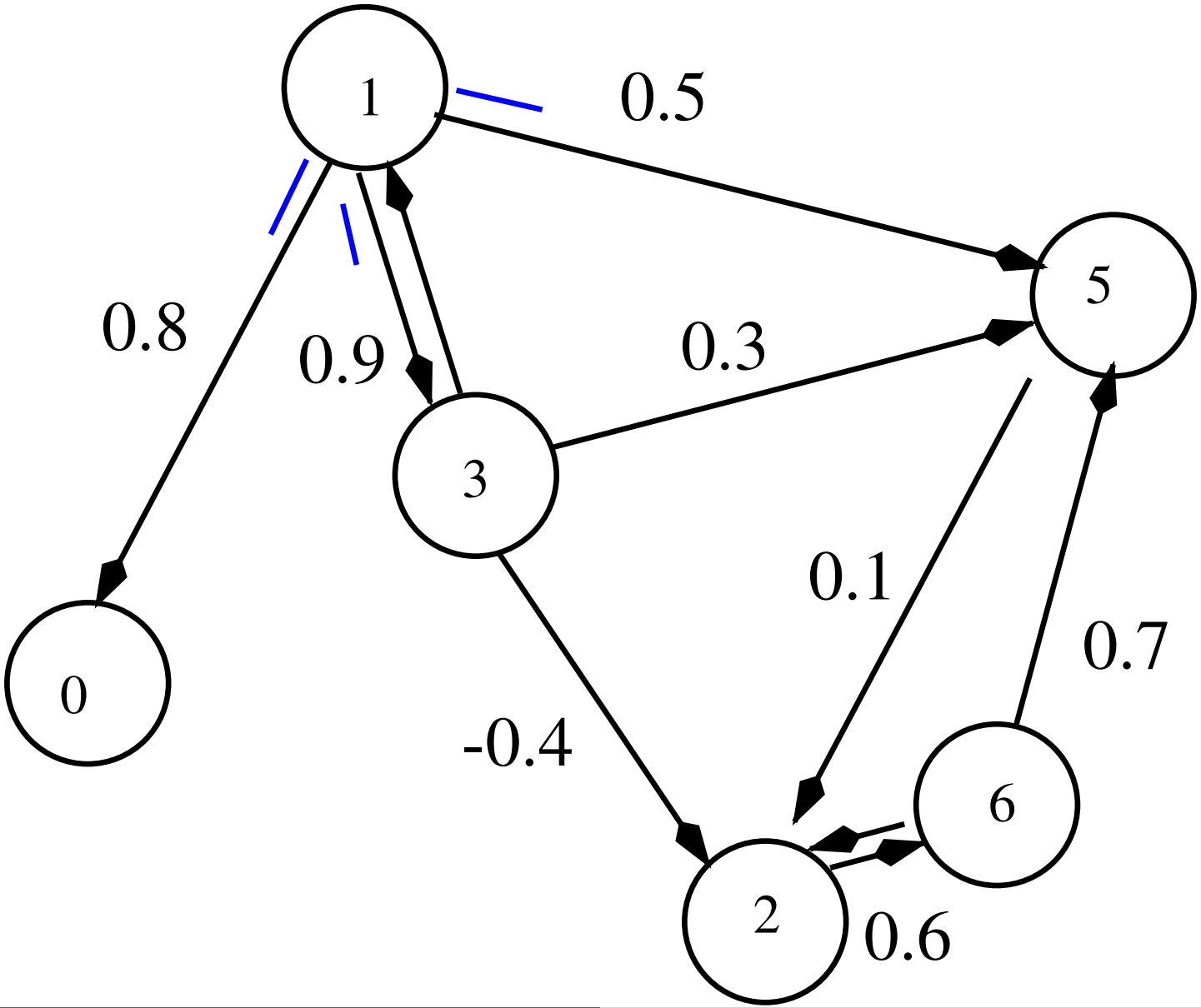


Ciclos negativos

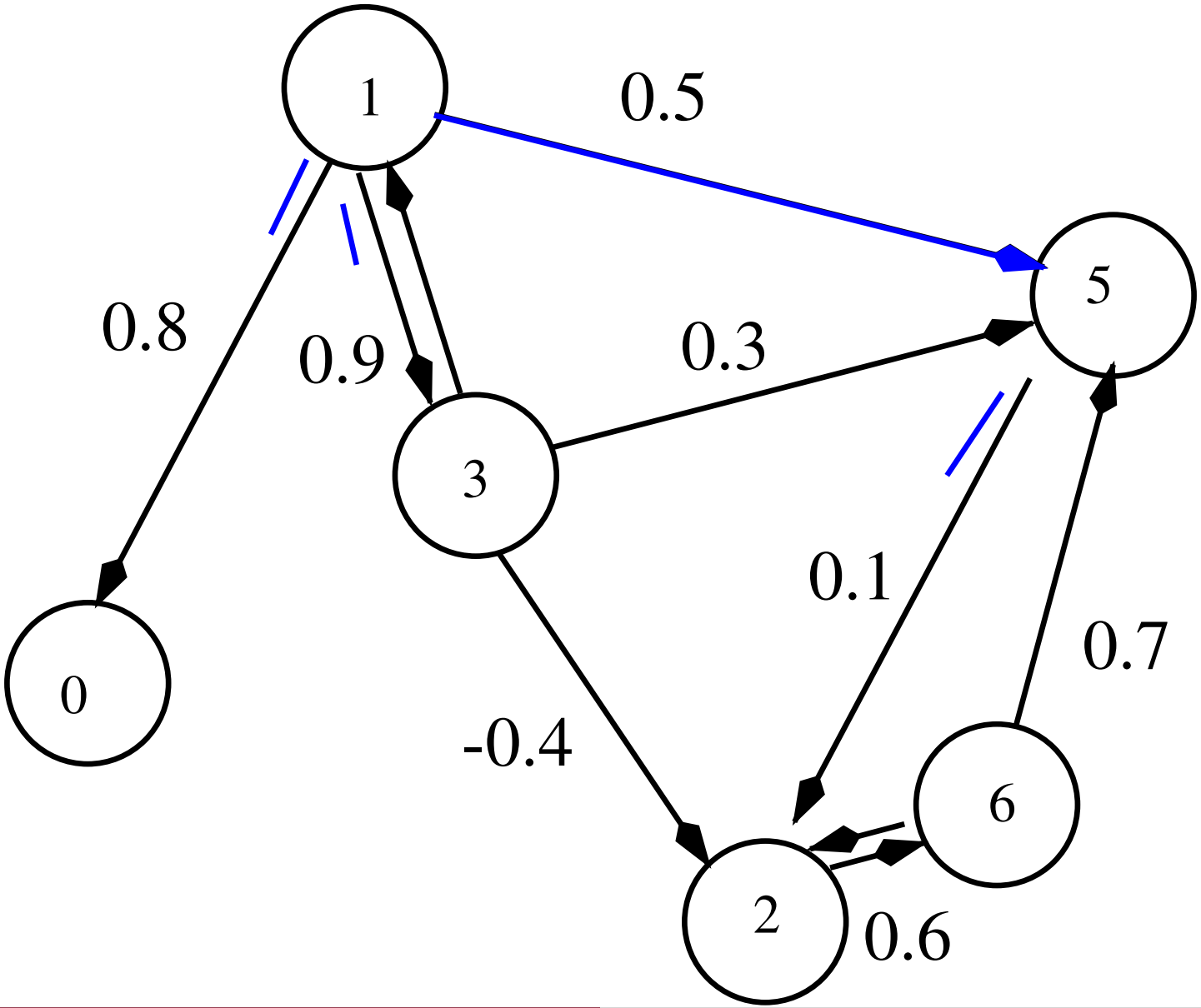
Propiedad: el problema de *job scheduling* con *deadlines* se reduce al problema de búsqueda de caminos mas cortos en grafos sin ciclos negativos.

- el primer problema que resolver es eso: buscar ciclos negativos; es un problema que no hay necesariamente ver del lado negativo (ejemplo: la especulación monetaria!)
- otro elemento es que Dijsktra tiene una naturaleza greedy (glotona) en ese caso y no funciona; que tal de Floyd?

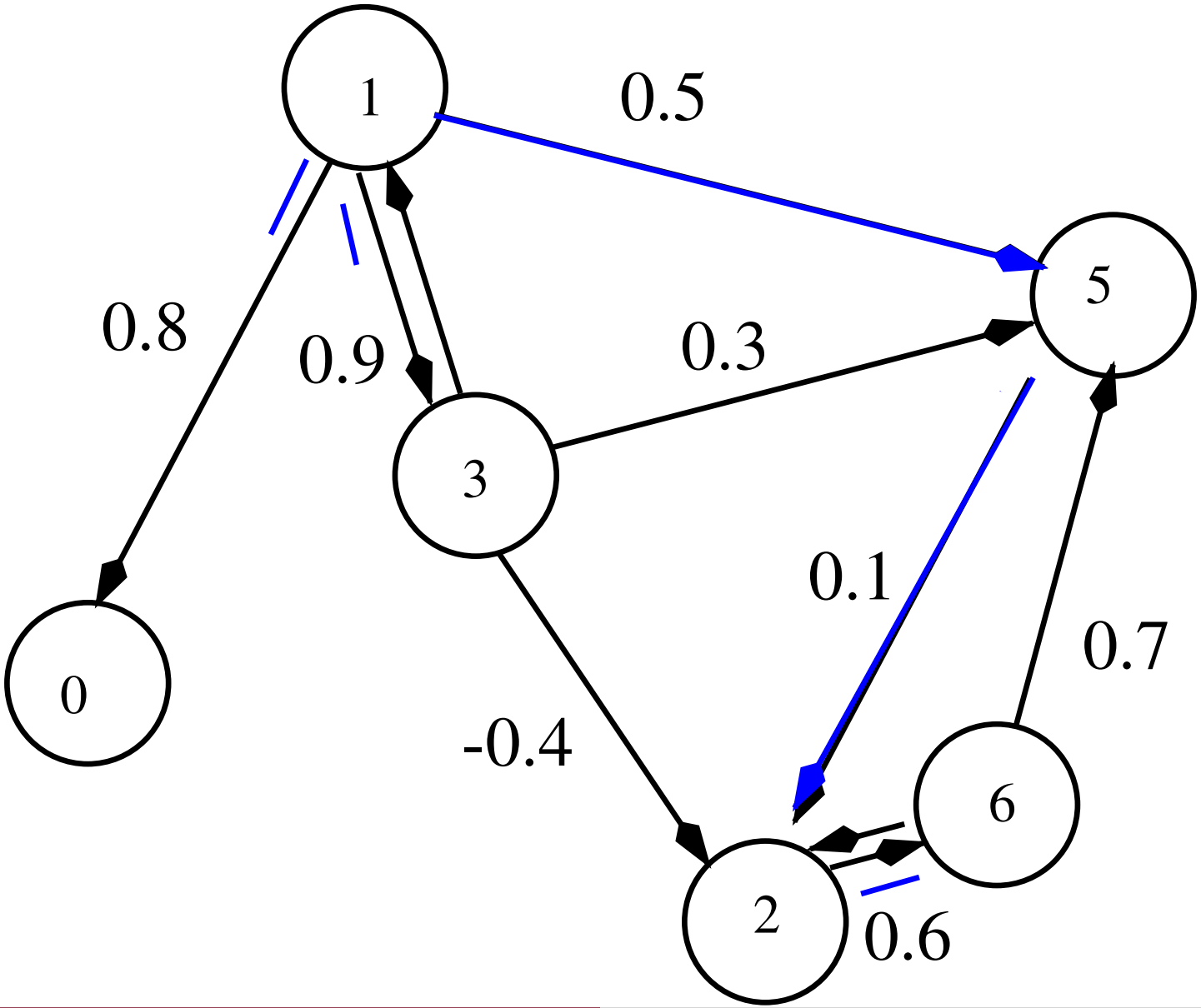
Dijsktra y los negativos



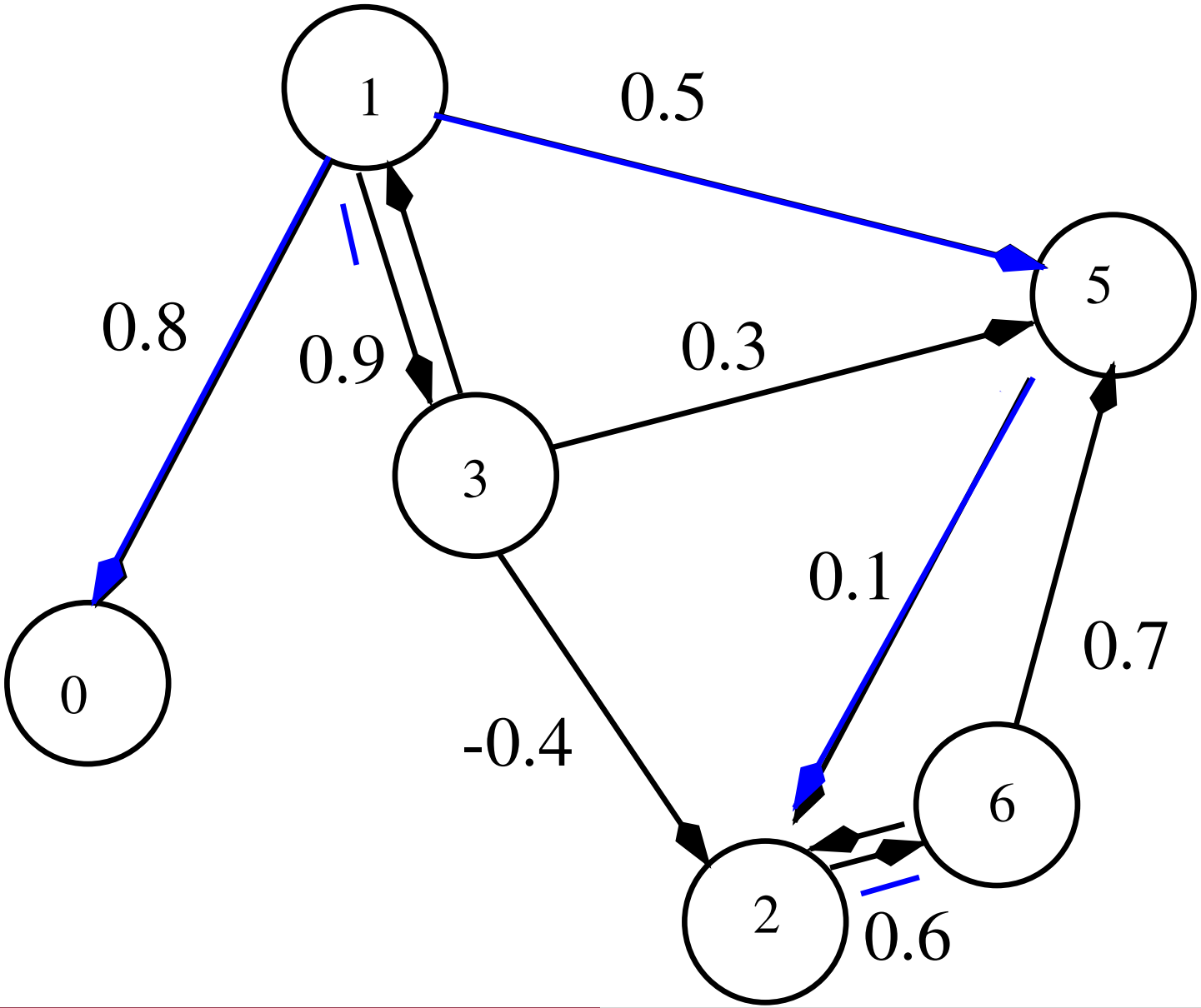
Dijsktra y los negativos



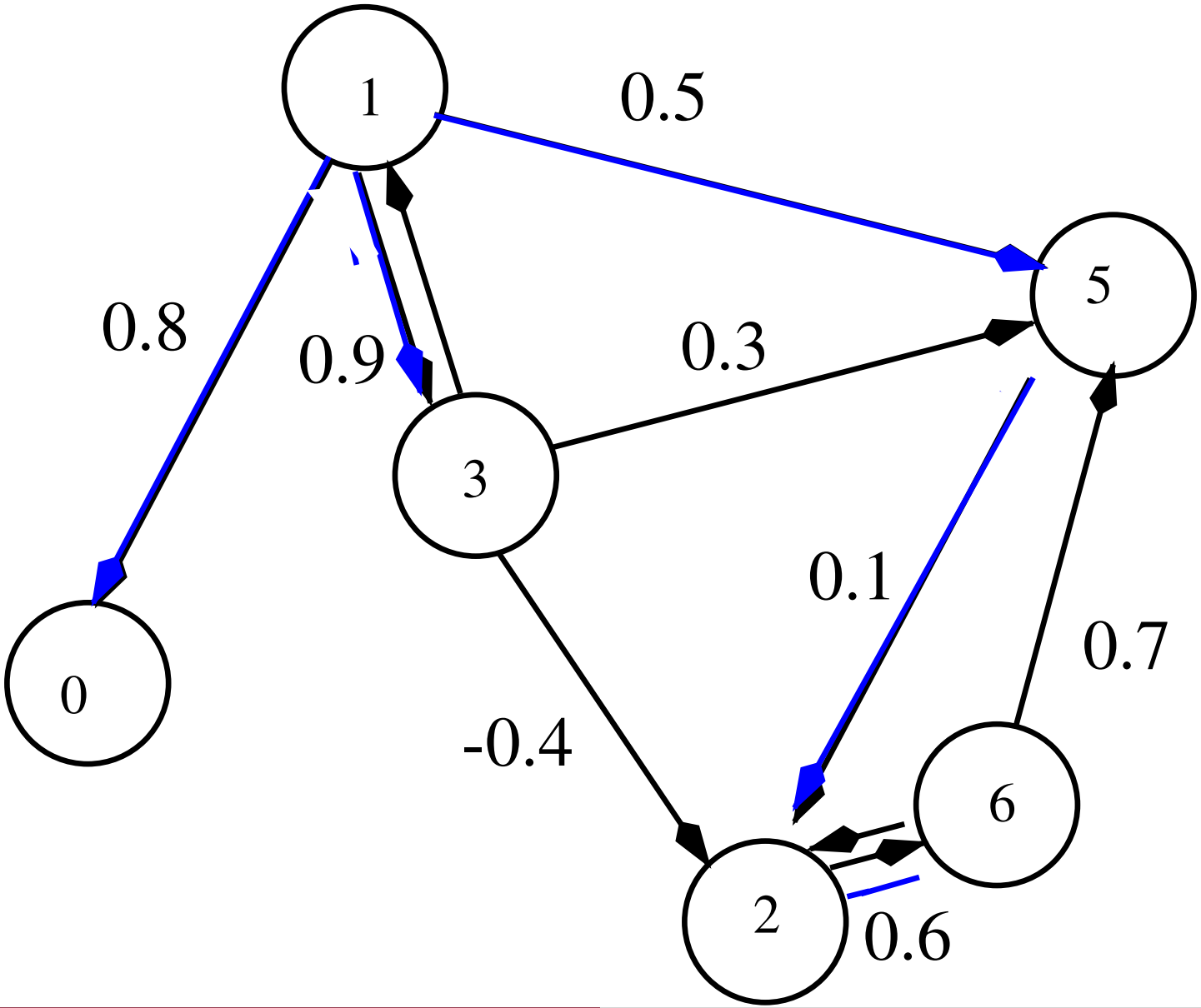
Dijsktra y los negativos



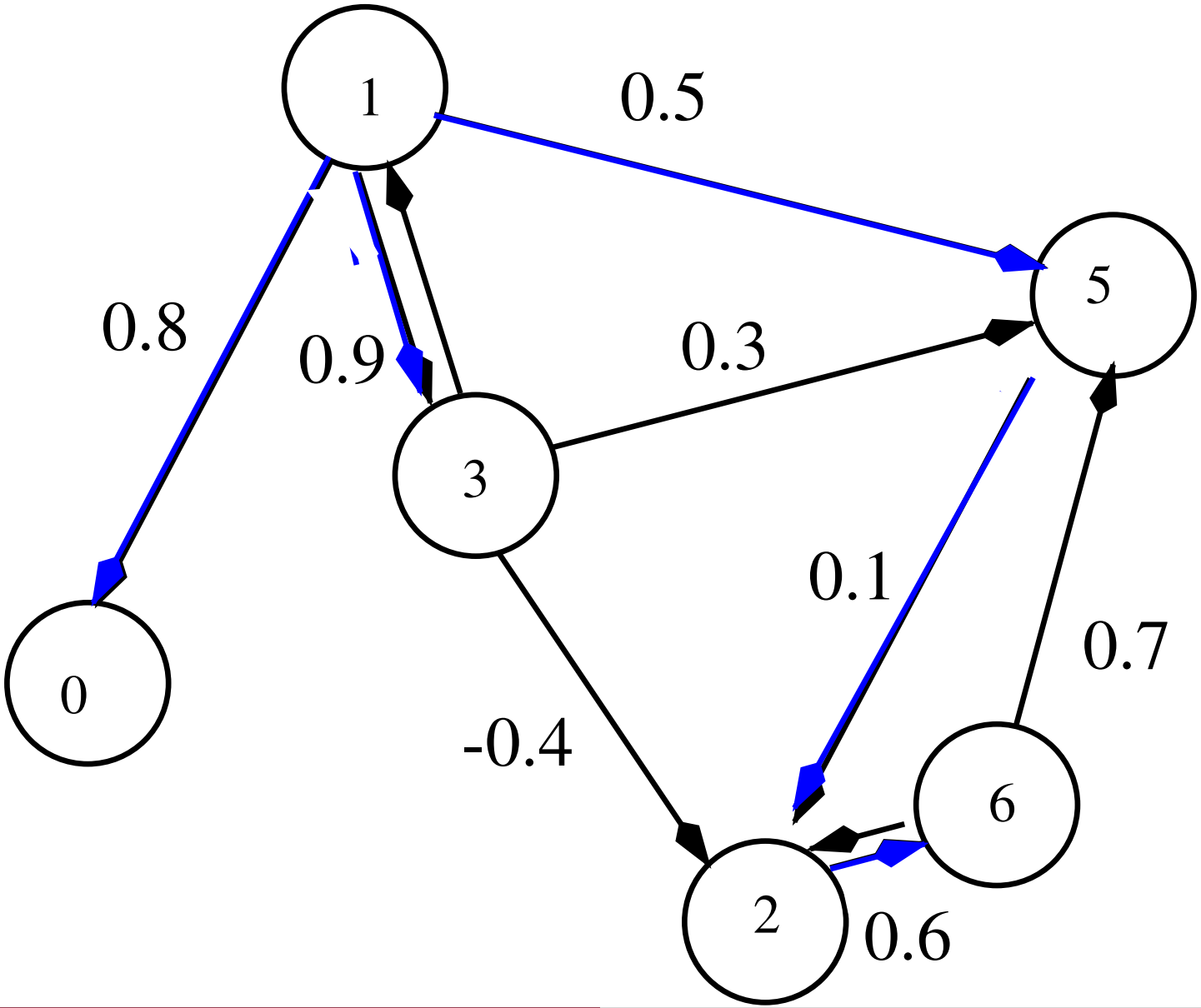
Dijsktra y los negativos



Dijsktra y los negativos



Dijsktra y los negativos



Floyd y los negativos

Floyd funciona correctamente en los casos en que no hay ciclos negativos (ver la prueba, no depende del signo de las ponderaciones) y esta capaz de detectar un ciclo negativo

- las entradas de la matriz de tamaños dicen que el algoritmo encontró un camino pasando por cada de los vértices al máximo una vez del tamaño el valor de la casilla
- en particular, una entrada diagonal en la matriz corresponde a al menos un ciclo negativo! pero no se puede decir mucho mas
- Algoritmo "eficiente" para resolver problemas de caminos mas cortos en grafos con ponderaciones negativas (pero sin ciclos negativos?). Dijsktra era $|A| \log |V|$ (para una fuente) o $|V||A| \log |V|$ (para todos pares)

Bellman-Ford

Idea para resolver el problema mono-fuente (s):

- Inicializar $wt[s] = 0$ y $wt[v] = \infty$ para los otros v
- Hacer $|V|$ pasos de relajación sobre todos los arcos visitados en cualquier orden...

Implementación ingenua

```
for (int n=0;n<V;n++)
 for (int v=0;v<V;v++) {
   typename Graph::adjIterator A(G, v);
   for (Edge* e = A.beg(); !A.end(); e = A.nxt()) {
          if (v!=s && spt[v]==NULL) continue;
          int w = e->w();
          double P = wt[v] + e->wt();
          if (P < wt[w]) { wt[w] = P; spt[w] = e; }
        }
}
```

Bellman-Ford

Prueba: la complejidad es $O(|V||A|)$; mostrar por inducción que (i) los caminos de tamaño $i+1$ aparecen a la iteracion i y que (ii) después de la iteración i, para todo vértice v, $wt[v]$ no es mayor que el tamaño del camino mas corto que contiene al maximo $i+1$ arcos

- después de $i = 0$ es obvio: la relajación solo ha podido añadir caminos mas cortos $s - v$
- si se supone ok a i, entonces después de la $i+1$, entre los caminos que contengan al máximo $i+2$ arcos, dos posibilidades: (1) el mínimo de ellos tiene un numero de arcos inferior a $i+1$, en este caso $wt[v]$ no cambia; (2) si el mínimo tiene $i+2$ arcos, consiste de dos partes $s - w - v$, la primera siendo minorada por el $wt[w]$ de antes, el $wt[v]$ sigue minorante entonces del camino mas corto; como, al final, nos paramos despues de $|V| - 1$ iteraciones (no ciclo) $wt[v]$ es a la vez minorante del camino mas corto y es el mismo un camino.

Bellman-Ford

Implementacion practica:

- Los únicos arcos interesantes en casa paso son los por los cuales el vértice de inicio ha cambiado en el paso anterior
- Implementación posible con una cola que almacena esos arcos
- Complejidad en el peor caso de $|A||V|$ (no tan chido)

Bellman-Ford

```
SPT(Graph &G, int s) : G(G),
   spt(G.V()), wt(G.V(), G.V())
{ QUEUE<int> Q; int N = 0;
   wt[s] = 0.0;
   Q.put(s); Q.put(G.V());
   while (!Q.empty())
   { int v;
      while ((v = Q.get()) == G.V())
         { if (N++ > G.V()) return; Q.put(G.V()); }
      typename Graph::adjIterator A(G, v);
      for (Edge* e = A.beg(); !A.end(); e = A.nxt())
         { int w = e->w();
            double P = wt[v] + e->wt();
            if (P < wt[w])
               { wt[w] = P; Q.put(w); spt[w] = e; }
         }
```

Bellman-Ford

- Detección de ciclos negativos: una idea?

Bellman-Ford

- Detección de ciclos negativos: una idea?
- Iterar una vez mas (una $|V|$-sima), si se hace una relajacion, es que hay un ciclo negativo: es un camino de $|V|$ vértices, y como el *wt* va decreciendo estrictamente, en el punto *w* del ciclo, entra una segunda vez con un *wt* inferior a lo que tenia la primera vez

Bellman-Ford

- Detección de ciclos negativos: una idea?
- Iterar una vez mas (una $|V|$-sima), si se hace una relajacion, es que hay un ciclo negativo: es un camino de $|V|$ vértices, y como el *wt* va decreciendo estrictamente, en el punto w del ciclo, entra una segunda vez con un *wt* inferior a lo que tenia la primera vez
- Algoritmo en $|A||V|$ para la detección de ciclos negativos

Bellman-Ford

Todos pares de vértices?

- Re-lanzar el Bellman-Ford desde todos vértices; solo se hace interesante para grafos escasos ($O(|V|^2|A|)$)
- Hay un principio que podemos usar: la re-ponderación

```
for (in v=0;v<V;c++) {
    typename Graph::adjIterator A(G, v);
    for (Edge* e = A.beg(); !A.end(); e = A.nxt())
     {
          e->wt() = e->wt() + wt[v] - wt[e->w()];
     }
}
```

Johnson

Propiedad: después de esta re-ponderación con el resultado de los caminos mas cortos a partir de un vértice s, tenemos ponderaciones positivas.

De ahí el algoritmo de Johnson:

1. Aplicar el algoritmo de Bellman-Ford
2. Si hay ciclo negativo, stop
3. Sino, re-ponderar
4. Aplicar Dijsktra en su versión "todos pares"

Johnson

Prueba de la propiedad: dado un arco $v - w$, despues de la primera etapa, salimos el camino mas corto a w y el camino mas corto a v. Si el primero incluye el segundo, entonces incluye también el arco $v - w$ y tenemos:

$$(wt[w] == wt[v] + e- > wt())$$

entonces un nuevo peso nulo, y sino el camino mas corto a w es de tamaño inferior a éste:

$$(wt[w] < wt[v] + e- > wt())$$

o sea, nuevo peso positivo
Complejidad en $|V||A| \log |V|$

Resumen de hasta ahora

Problemas que hemos encontrado:

- *Grafos no dirigidos*:

Búsqueda de camino entre v y w	DFS	$O(\|V\| + \|A\|)$
Arbol o bosque?	DFS	$O(\|V\| + \|A\|)$
Componentes conectas	DFS	$O(\|V\| + \|A\|)$
Camino hamiltoniano	Recursivo	$O((\|V\| - 2)!)$
Camino euleriano	Test grado	$O(\|V\|)$
2-Colorización	DFS	$O(\|V\| + \|A\|)$
Puentes, puntos de articulación	DFS	$O(\|V\| + \|A\|)$
Componentes biconectadas	DFS	$O(\|V\| + \|A\|)$
Camino mas corto desde v	BFS	$O(\|V\| + \|A\|)$

Resumen de hasta ahora

- *Grafos dirigidos*:

DAG?	DFS	$O(\|V\| + \|A\|)$
Cerradura transitiva	DFSs	$O(\|V\|(\|V\| + \|A\|))$
Cerradura transitiva	Warshall	$O(\|V\|^3)$
Conectividad fuerte	2 DFSs (Kosaraju)	$O(\|V\| + \|A\|)$
Conectividad fuerte	DFS (Tarjan, Gabow)	$O(\|V\| + \|A\|)$

- *Grafos dirigidos aciclicos*:

Ordenamiento topologico	DFS (postorden)	$O(\|V\| + \|A\|)$
Cerradura transitiva	Lineas	$O(\|V\|\|A\|)$
Cerradura transitiva	DFSs	$O(\|V\|^2 + \|V\|X)$

Resumen de hasta ahora

- *Grafos ponderados*:

AGM	PFS (Prim)	$O(\|V\|^2)$
AGM	Cola de prioridad (Prim)	$O(\|A\| \log \|V\|)$
AGM	Kruskal	$O(\|A\| \log \|A\|)$
AGM	Boruvka	$O(\|A\| \log \|V\|)$

- *Grafos dirigidos ponderados*:

C.m.c. desde v	PFS (Dijsktra)	$O(\|V\|^2)$
C.m.c. desde v	Cola de prioridad (Dijsktra)	$O(\|A\| \log \|V\|)$
Todos c.m.c.	Dijsktra	$O(\|V\|\|A\| \log \|V\|)$
Todos c.m.c.	Floyd	$O(\|V\|^3)$

Resumen de hasta ahora

- *Grafos dirigidos ponderados aciclicos*:

C.m.c. o C.m.l. desde $\{v\}_i$	Ordenamiento top.	$O(\|V\| + \|A\|)$
Todos C.m.c	DFS	$O(\|V\|\|A\|)$

- *Grafos dirigidos ponderados de pesos negativos*:

C.m.c. desde v	Bellman -Ford	$O(\|V\|\|A\|)$
Todos c.m.c.	Floyd	$O(\|V\|^3)$
Todos c.m.c.	Bellman -Ford	$O(\|V\|^2\|A\|)$
Todos c.m.c.	Johnson	$O(\|V\|\|A\| \log \|V\|)$