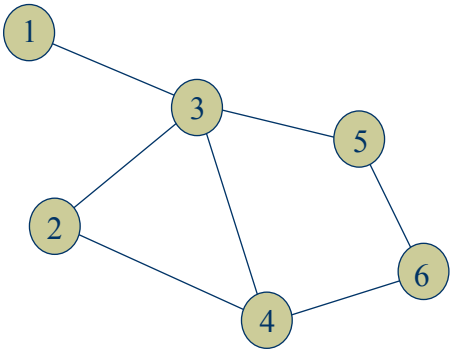
El Arte de la Programación Rápida

Programación Dinámica

Problema ejemplo

- Vamos a analizar la técnica de Programación dinámica a través de un ejemplo.
- Calcular los caminos más cortos entre **todos los pares** de nodos de un grafo.

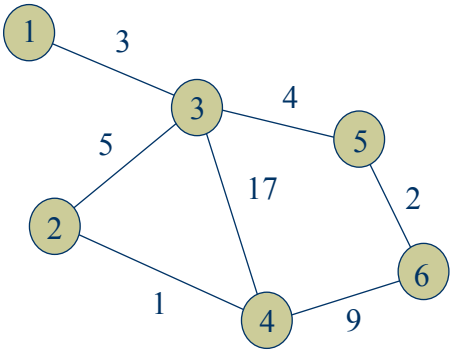
Problema ejemplo



	1	2	3	4	5	6
1	0	2	1	2	2	3
2	2	0	1	1	2	2
3	1	1	0	1	1	2
4	2	1	1	0	2	1
5	2	2	1	2	0	1
6	3	2	2	1	1	0

Medimos longitud de caminos

Problema ejemplo

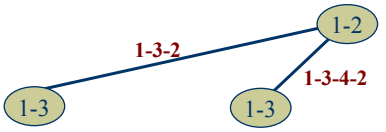


	1	2	3	4	5	6
1	0	8	3	9	7	9
2	8	0	5	1	9	10
3	3	5	0	6	4	6
4	9	1	6	0	10	9
5	7	9	4	10	0	2
6	9	10	6	9	2	0

Medimos costo de caminos

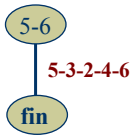
Primera aproximación: *backtracking*

- No, no otra vez!!!! ;)
- ¿Cuál es el espacio de búsqueda?



...

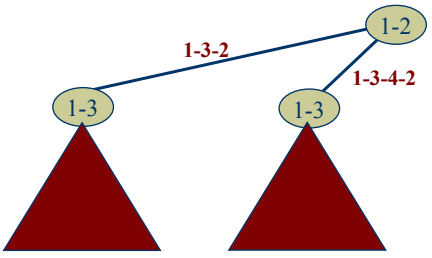
...



Cada camino desde 1-2 a fin me da una combinación de caminos desde cada nodo a todos los demás.

Primera aproximación : *backtracking*

- Funciona, pero esta solución es muy ineficiente.
- Rápidamente se puede ver que, aún si el grafo es no dirigido, el algoritmo es proporcional a n!
  - Para n=10, ya tengo un orden de 10^6 elementos que procesar...
- Problema: vuelvo a calcular lo mismo una y otra vez:



¡Los subárboles son iguales!

Idea: recordar resultados parciales

Programación Dinámica:

técnica para implementar eficientemente algoritmos recursivos por medio del almacenamiento de resultados intermedios

Caminos más cortos con P.D.

- Ordeno (numero) los nodos
  - (ya estaba hecho ☺)
  - pienso algoritmo recursivo:
    - En cada nivel de recursión *k*, considero todos los caminos que pasan solo por nodos con índice menor que *k*.
    - De hecho, ni siquiera necesitamos que sea recursivo!

Caminos más cortos con P.D.

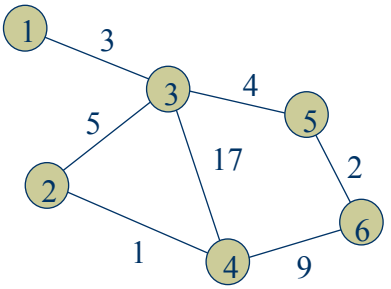
- Algoritmo:
  - Para cada nodo intermedio *k* (entre 1 y *n*)
    - Para cada nodo de salida *i*
      - Para cada nodo de llegada *j*
        - Veo si utilizando el nodo *k* existe un camino más corto que el camino que ya existía entre *i* y *j*.
        - Si existe, lo apunto, si no, me quedo con el anterior

Caminos más cortos con P.D.

- Implementación:
  - Utilizamos matriz de adyacencia
    - Implica utilizar espacio de n^2, pero al fin y al cabo tenemos que guardar la información de distancia entre nodos de todas formas.

costo

0	∞	3	∞	∞	∞
∞	0	5	1	∞	∞
3	5	0	17	4	∞
∞	1	17	0	∞	9
∞	∞	4	∞	0	2
∞	∞	∞	9	2	0



Costo de caminos más cortos

- Algoritmo:

```
for (k=1; k<=n; k++)
  for (i=1; i<=n; i++)
     for (j=1; j<=n; j++){
          nuevo = costo(i,k)+costo(k,j);
          if (nuevo < costo(i,j))
               costo(i,j) = nuevo;
     }
```


Caminos más cortos

- El algoritmo anterior me da el menor costo.
- Pero no me dice el camino que tiene asociado dicho costo.
- Para saber el camino, basta guardar el nodo intermedio de cada paso.

Caminos más cortos con P.D.

- Algoritmo:

```
for (k=1; k<=n; k++)
  for (i=1; i<=n; i++)
     for (j=1; j<=n; j++){
          nuevo = costo(i,k)+costo(k,j);
          if (nuevo < costo(i,j)){
               costo(i,j) = nuevo;
               camino(i,j) = k;
          }
     }
```


¿Cuál es el camino?

- Si queremos imprimir el camino de mínimo costo entre x e y:

```
Hacemos z = camino(x,y).
Si z==y, imprimimos y //ya hemos llegado.
Sino, imprimimos camino de x a z y de z a y.
```