Cómo implementar eficientemente el algoritmo de Dijkstra?

La idea clave será utilizar un priority queue ó heap. Esta es una estructura de datos particular que interactúa muy bien con Dijkstra.

Obs: La interacción entre algoritmos y estructuras de datos es una idea clave para lograr programas eficientes. Para ello hay que tener familiaridad con varias estructuras de datos así que empezaremos por discutir las ideas generales sobre heaps.

Def: Un heap o priority queue es como un diccionario {Keys : Values} donde las Keys son elementos de un conjunto totalmente ordenado (por ejemplo los enteros). Un heap H tiene algunas operaciones MUY EFICIENTES:

(1) INSERT (k, v): Introduce una nueva pareja (k, v) en H.

(2) ExtractMIN: Encuentra en H una pareja (k^*, v^*) donde k^* es el MÍNIMO de las keys de H en el orden total. Retorna (k^*, v^*) y lo elimina de H. (Si el mínimo se alcanza en varias parejas retorna y quita cualquiera de ellas)

(3) DELETE: Dado un apuntador a un objeto (k, v) en H, lo quita de H.

Teorema: Existe una manera H de representar los datos que permite realizar las operaciones (1) y (2) en tiempo $O(\log(n))$ donde n es el número de parejas en H

Al final explicaremos cómo, por ahora usémoslas...

EN PYTHON:

```
import heapq
H = []
heapq.heappush(H, (1, "definición"))
heapq.heappop(H)
```

INSERT

EXTRACT MIN de este key

¿Cuándo se usa un heap?

"Cuando queremos incluir objetos (en un orden cualquiera) y extraerlos en orden creciente (es decir, el más bajito primero)"

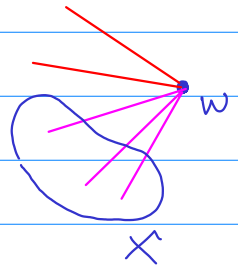
El ejemplo más sencillo es un "event_handler" por ejemplo al programar un juego en el que registramos el instante (key) en que deben ocurrir una serie de eventos (value) y usamos el heap para saber cuál es el más próximo a ser procesado ← EXTRACT-MIN.

Dijkstra con heap: [Usaremos la notación de Dijkstra la clase anterior...]

Idea: Mantenemos un heap que recuerde los vértices de $V(G) \setminus X$ donde

value(w) = "índice del vértice $w \in V(G)$"

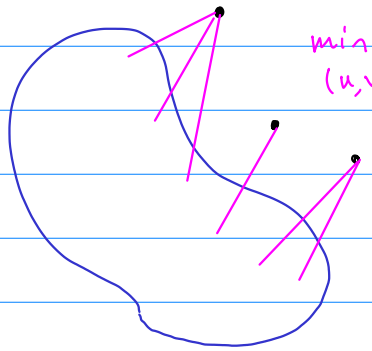
$$(*) \quad \left[\text{key}(w) = \min_{u \in In(w) \cap X} \{ \ell(u,w) + \varphi(u) \} \right]$$



$w^* = \text{EXTRACT\_MIN}(H)$, $X \leftarrow X \cup \{w^*\}$, $\varphi(w^*) = \text{key}(w^*)$ (P)

Lema: Si los keys de H satisfacen la ecuación (*) entonces (P) lleva al siguiente paso correcto de Dijkstra

Dem: Este paso puede parecer sorprendente pues Dijkstra busca el mínimo entre las aristas que cruzan, pero



$$\min_{(u,w):\, u \in X,\, w \notin X,\, (u,w) \in E(G)} \ell(u,w) + \varphi(u)$$

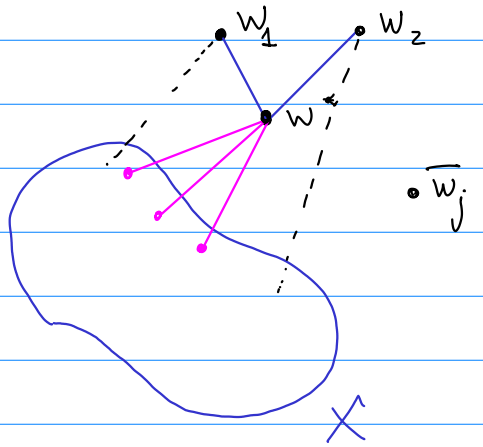
$$= \min_{w \in V \setminus X} \left[\min_{(u,w):\, u \in X,\, (u,w) \in E(G)} \ell(u,w) + \varphi(u) \right]$$

$$= \min_{w \in V \setminus X} \text{key}(w)$$

demostrado el Lema ●

El problema es que, después de incluir w^* en X la ecuación (*) puede fallar en algunos vértices PORQUE X CAMBIÓ y debemos

ajustar nuestro heap para que la ecuación (*) se cumpla para el nuevo conjunto X.
Para ello debemos entender donde puede haber cambios:



Si $\neg\exists\,(w^*, \overline{w}_j) \Rightarrow key(\overline{w}_j)$ OK ✓

Si $\exists\,(w^*, w_i) \Rightarrow key(w_i) = \min\{key(w_i),\ \ell(w^*, w_i) + \varphi(w^*)\}$

mejor resultado con aristas existentes

nuevo candidato por el mínimo

$Q_t(w^*) \cap [X \setminus V(G)]$

¿Cuántos llamados a las operaciones del Heap hacemos?

EXTRACT_MIN — 1 vez cada vértice

DELETE e INSERT — máximo una vez por arista

Teorema: [Dijkstra con heaps] Si G es un grafo cualquiera, $s \in V(G)$ y $\ell \geq 0$ es una colección de pesos positivos enteros todos los valores $d(s, v)$, $v \in V(G)$ pueden calcularse en $O((m+n)\log(n))$

Obs: Esta implementación es MUY eficiente y puede correr en grafos de millones de vértices en un laptop.