

Welcome to Algorithms and Data Structures! - CS2100

Algoritmos voraces, golosos, codiciosos (Greedy algorithms)

1. Qué es?

Siempre busca la mejor solución local, esperando tener la mejor solución global

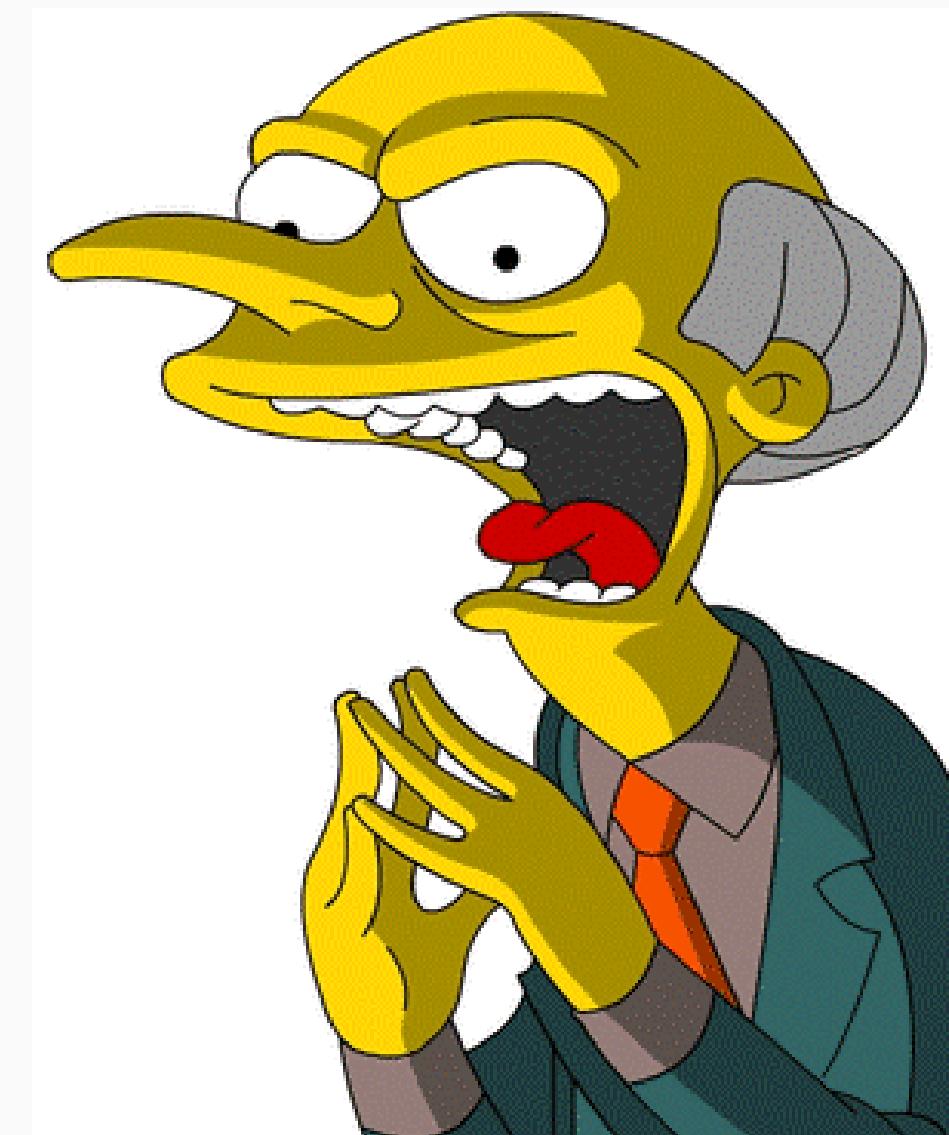
2. Cuál es el problema de esto?

Se ignora el efecto a futuro.
No necesariamente llega a la solución optima global

3. Algoritmos voraces tienen dos propiedades

- Subestructuras óptimas
- Elección codiciosa

Para muchos problemas, utilizar un algoritmo voraz puede fallar. **Entonces en qué casos se podría utilizar?**



Algoritmos voraces (esquema genérico)

```
función voraz (C:conjunto)
    devuelve conjunto
    {C es el conjunto de todos los candidatos}
principio
    S:=∅; {S es el conjunto en el que se
            construye la solución}
    mq ¬solución(S) ∧ C≠∅ hacer
        x:=elemento de C que
            maximiza seleccionar(x);
        C:=C-{x};
        si completable(S∪{x})
            entonces S:=S∪{x}
        fsi
    fmq;
    si solución(S)
        entonces devuelve S
        sino devuelve no hay solución
    fsi
fin
```

Algoritmos voraces, golosos, codiciosos (Greedy algorithms)

**Como implementamos PRIM y KRUSKAL
con algoritmos voraces?**

Kruskal

```
función voraz(C:conjunto)
    devuelve conjunto
{C es el conjunto de todos los candidatos}
principio
    S:=∅; {S es el conjunto en el que se
            construye la solución}
    mq ¬solución(S) ∧ C≠∅ hacer
        x:=elemento de C que
            maximiza seleccionar(x);
        C:=C-{x};
        si completable(S ∪ {x})
            entonces S:=S ∪ {x}
        fsi
    fmq;
    si solución(S)
        entonces devuelve S
        sino devuelve no hay solución
    fsi
fin
```

Funcion kruskal(Graph(V,E))

Solución(S): árbol que cubre todos los vértices
Seleccionar(C): extraer arista con peso mínimo de C
Completable(X, S): verificar que X no forme ciclo en S

Kruskal

```
función voraz(C:conjunto)
    devuelve conjunto
{C es el conjunto de todos los candidatos}
principio
    S:=∅; {S es el conjunto en el que se
            construye la solución}
    mq ¬solución(S) ∧ C≠∅ hacer
        x:=elemento de C que
            maximiza seleccionar(x);
        C:=C-{x};
        si completable(S∪{x})
            entonces S:=S∪{x}
        fsi
    fmq;
    si solución(S)
        entonces devuelve S
        sino devuelve no hay solución
    fsi
fin
```

Funcion kruskal(Graph(V,E))

- C = MinHeap(E)
- S = {} //árbol
- While ¬Solución(S,V) and |C| != 0
 - X = C.getMin()
 - C.pop()
 - If Completable(X, S):
 - S = S + X
- Devolver S

Solución(S): $|S| == |V|$ árbol que cubre todos los vértices
Seleccionar(C): extraer arista con peso mínimo de C
Completable(X, S): verificar que X no forme ciclo en S

PRIM

```
función voraz(C:conjunto)
    devuelve conjunto
{C es el conjunto de todos los candidatos}
principio
    S:=∅; {S es el conjunto en el que se
            construye la solución}
    mq ¬solución(S) ∧ C≠∅ hacer
        x:=elemento de C que
            maximiza seleccionar(x);
        C:=C-{x};
        si completable(S∪{x})
            entonces S:=S∪{x}
        fsi
    fmq;
    si solución(S)
        entonces devuelve S
    sino devuelve no hay solución
    fsi
fin
```

- Funcion PRIM(Graph(V,E))

Solución(S): $|S| == |V|$

Seleccionar(C): devolver un arista adyacente a $u \in S$ con el menor peso

Completable(a, T): Si a no forma ciclo en T : Siempre verdadero

```

función voraz (C:conjunto)
    devuelve conjunto
    {C es el conjunto de todos los candidatos}
principio
    S:=∅; {S es el conjunto en el que se
            construye la solución}
    mq ¬solución(S) ∧ C≠∅ hacer
        x:=elemento de C que
            maximiza seleccionar(x);
        C:=C-{x};
        si completable(S ∪ {x})
            entonces S:=S ∪ {x}
        fsi
    fmq;
    si solución(S)
        entonces devuelve S
        sino devuelve no hay solución
    fsi
fin

```

- Funcion PRIM(Graph(V,E), v)

- C = MinHeap(E incidentes al vértice v)
- S = {v} //vertices visitados
- T = {} // árbol solución
- While ¬Solucion(S,V) and |C| != 0:
 - {u, v} = Seleccionar(S, C)
 - C = C - {u, v}
 - If Completable(v, T)
 - S = S ∪ v
 - T = T ∪ {u, v}
 - C = C ∪ {aristas incidentes a T}

Solución(S): $|S| == |V|$

Seleccionar(C): devolver un arista adyacente a u de S con el menor peso

Completable(a, T): Si a no forma ciclo en T, u y v visitados

Welcome to Algorithms and Data Structures! - CS2100