Algoritmos voraces

José de Jesús Lavalle Martínez

Benemérita Universidad Autónoma de Puebla
Facultad de Ciencias de la Computación
Maestría en Ciencias de la Computación
Análisis y Diseño de Algoritmos
MCOM 20300

Contenido

Introducción I

- Los algoritmos voraces son la primera familia de algoritmos que examinamos en detalle, la razón es simple: generalmente son los más sencillos.

Introducción I

- Como sugiere su nombre, su enfoque es miope y toman decisiones sobre la base de la información que tienen a mano sin preocuparse por el efecto que estas decisiones puedan tener en el futuro.

Introducción I

- Por lo tanto, son fáciles de inventar, fáciles de implementar y, cuando funcionan, son eficientes.

Introducción I

- Sin embargo, dado que el mundo rara vez es tan simple, muchos problemas no pueden resolverse correctamente con un enfoque tan crudo.

Introducción II

- Los algoritmos voraces se utilizan normalmente para resolver problemas de optimización.

Introducción II

- Los ejemplos que veremos incluyen encontrar la ruta más corta de un nodo a otro a través de una red, o encontrar el mejor orden para ejecutar un conjunto de trabajos en una computadora.

Introducción II

- En tal contexto, un algoritmo voraz funciona eligiendo el arco, o el trabajo, que parece más prometedor en cualquier momento; nunca reconsidera esta decisión, cualquiera que sea la situación que pueda surgir posteriormente.

Introducción II

- No es necesario evaluar alternativas ni emplear procedimientos de contabilidad elaborados que permitan deshacer decisiones anteriores. Comenzamos con un ejemplo cotidiano en el que esta táctica funciona bien.

Ejemplo 1 “Dar cambio” I

- Supongamos que vivimos en un país donde están disponibles las siguientes monedas: dólares (100 centavos), cuartos (25 centavos), dieces (10 centavos), cincos (5 centavos) y centavos (1 centavo).

Ejemplo 1 “Dar cambio” I

- Nuestro problema es diseñar un algoritmo para pagar una cantidad determinada a un cliente utilizando la menor cantidad posible de monedas.

Ejemplo 1 “Dar cambio” I

- Por ejemplo, si debemos pagar \$2.89 (289 centavos), la mejor solución es darle al cliente 10 monedas: 2 dólares, 3 cuartos, 1 diez y 4 centavos.

Ejemplo 1 “Dar cambio” I

- La mayoría de nosotros resolvemos este tipo de problemas todos los días sin pensarlo dos veces, inconscientemente usando un algoritmo voraz obvio: comenzando con nada, en cada etapa agregamos a las monedas ya elegidas una moneda del mayor valor disponible que no sobrepase la cantidad a pagar.

Ejemplo 1 “Dar cambio” II

El algoritmo se puede formalizar como sigue.

function MAKE-CHANGE(n)

```
1   const C = {100, 25, 10, 5, 1}
2   S ← ∅
3   s ← 0
4   while s ≠ n
5   do x ←  el elemento más grande en C tal que s + x ≤ n
6       if  no existe tal elemento
7          then return  “no se encontró solución”
8       S ← S ∪ { una moneda de valor x }
9       s ← s + x
10  return S
```

Ejemplo 1 “Dar cambio” III

- Es fácil convencerse a uno mismo (pero sorprendentemente difícil de probar formalmente) de que con los valores dados para las monedas, y siempre que se disponga de un suministro adecuado de cada denominación, este algoritmo siempre produce una solución óptima a nuestro problema.

Ejemplo 1 “Dar cambio” III

- Sin embargo, con una serie de valores diferentes, o si el suministro de algunas de las monedas es limitado, es posible que el algoritmo voraz no funcione; consulte los problemas 6.2 y 6.4 del libro de texto.

Ejemplo 1 “Dar cambio” III

- En algunos casos, puede elegir un conjunto de monedas que no es óptimo (es decir, el conjunto contiene más monedas de las necesarias), mientras que en otros puede no encontrar una solución aunque exista (si bien esto no puede suceder si se tiene una cantidad ilimitada de monedas de 1 unidad).

Ejemplo 1 “Dar cambio” IV

- El algoritmo es “voraz” porque en cada paso elige la moneda más grande que puede, sin preocuparse de si esta será una decisión acertada a largo plazo.

Ejemplo 1 “Dar cambio” IV

- Además, nunca cambia de opinión: una vez que se ha incluido una moneda en la solución, ésta seguirá ahí para siempre.

Ejemplo 1 “Dar cambio” IV

- Como explicaremos más adelante, estas son las características de esta familia de algoritmos. Para el problema particular de dar cambio, en el capítulo 8 se describe un algoritmo completamente diferente. Este algoritmo alternativo usa programación dinámica.

Ejemplo 1 “Dar cambio” IV

- El algoritmo de programación dinámica siempre funciona, mientras que el algoritmo voraz puede fallar; sin embargo, es menos sencillo que el algoritmo voraz y (cuando ambos algoritmos funcionan) menos eficiente.

Características generales I

Por lo general, los algoritmos voraces y los problemas que pueden resolver se caracterizan por la mayoría o todas las características siguientes.

Características generales I

- Tenemos algún problema que solucionar de forma óptima. Para construir la solución de nuestro problema, tenemos un conjunto (o lista) de candidatos: las monedas que están disponibles, las aristas de un grafo que pueden usarse para construir una ruta, el conjunto de trabajos a programar, o lo que sea.

Características generales I

- A medida que avanza el algoritmo, acumulamos otros dos conjuntos. Uno contiene candidatos que ya han sido considerados y elegidos, mientras que el otro contiene candidatos que han sido considerados y rechazados.

Características generales II

- Existe una función que comprueba si un conjunto particular de candidatos proporciona una solución a nuestro problema, ignorando las cuestiones de optimización por el momento.

Características generales II

- Por ejemplo, ¿las monedas que hemos elegido se suman a la cantidad a pagar? ¿Las aristas seleccionadas proporcionan una ruta al nodo que deseamos alcanzar? ¿Se han programado todos los trabajos?

Características generales II

- Una segunda función comprueba si un conjunto de candidatos es factible, es decir, si es posible o no completar el conjunto añadiendo más candidatos para obtener al menos una solución a nuestro problema.

Características generales II

- Aquí tampoco nos preocupamos por el momento de la optimización. Por lo general, esperamos que el problema tenga al menos una solución que se pueda obtener utilizando candidatos del conjunto inicialmente disponible.

Características generales III

- Otra función más, la función de selección, indica en cualquier momento cuál de los candidatos restantes, que no han sido elegidos ni rechazados, es el más prometedor.

Características generales III

- Finalmente, una función objetivo da el valor de la solución que hemos encontrado: la cantidad de monedas que usamos para dar el cambio, la longitud del camino que construimos, el tiempo necesario para procesar todos los trabajos en el programa, o cualquier otro valor que estemos tratando de optimizar.

Características generales III

- A diferencia de las tres funciones mencionadas anteriormente, la función objetivo no aparece explícitamente en el algoritmo voraz.

Características generales IV

- Para resolver nuestro problema buscamos un conjunto de candidatos que constituya una solución, y que optimice (minimice o maximice, según sea el caso) el valor de la función objetivo. Un algoritmo voraz avanza paso a paso.

Características generales IV

- Inicialmente, el conjunto de candidatos elegidos está vacío.

Características generales IV

- Luego, en cada paso, consideramos agregar a este conjunto el mejor candidato no probado restante, y nuestra elección se guía por la función de selección.

Características generales IV

- Si el conjunto ampliado de candidatos elegidos ya no fuera factible, rechazamos al candidato que estamos considerando actualmente.

Características generales IV

- En este caso, el candidato que ha sido juzgado y rechazado nunca se vuelve a considerar.

Características generales V

- Sin embargo, si el conjunto ampliado aún es factible, agregamos el candidato actual al conjunto de candidatos elegidos, donde permanecerá de ahora en adelante.

Características generales V

- Cada vez que ampliamos el conjunto de candidatos elegidos, comprobamos si ahora constituye una solución a nuestro problema.

Características generales V

- Cuando un algoritmo voraz funciona correctamente, la primera solución que se encuentra de esta manera siempre es óptima.

Características generales VI

```
function GREEDY(C : set)
 1   S ← ∅
 2   while C ≠ ∅ and not SOLUTION(S)
 3   do x ← SELECT(x)
 4        C ← C \ {x}
 5        if FEASIBLE(S ∪ {x})
 6          then S ← S ∪ {x}
 7   if SOLUTION(S)
 8     then return S
 9     else  return  "no hay soluciones"
```

Características generales VII

- Está claro por qué tales algoritmos se denominan “voraces”: en cada paso, el procedimiento elige el mejor bocado que puede tragar, sin preocuparse por el futuro.
- Nunca cambia de opinión: una vez que se incluye a un candidato en la solución, está ahí para siempre; una vez que se excluye a un candidato de la solución, nunca se reconsidera.
- La función de selección suele estar relacionada con la función objetivo.

Características generales VIII

- Por ejemplo, si estamos tratando de maximizar nuestras ganancias, es probable que elijamos al candidato restante que tenga el valor individual más alto.

Características generales VIII

- Si estamos tratando de minimizar el costo, entonces podemos seleccionar el candidato restante más barato, y así sucesivamente.

Características generales VIII

- Sin embargo, veremos que en ocasiones puede haber varias funciones de selección plausibles, por lo que tenemos que elegir la correcta si queremos que nuestro algoritmo funcione correctamente.

Características generales IX

Volviendo por un momento al ejemplo de dar cambio, aquí hay una forma en que las características generales de los algoritmos voraces pueden equipararse con las características particulares de este problema.

Características generales IX

- Los candidatos son un conjunto de monedas, representando en nuestro ejemplo 100, 25, 10, 5 y 1 unidades, con suficientes monedas de cada valor que nunca se nos acaban (sin embargo, el conjunto de candidatos debe ser finito).

Características generales IX

- La función de solución comprueba si el valor de las monedas elegidas hasta ahora es exactamente el importe a pagar.

Características generales IX

- Un juego de monedas es factible si su valor total no excede la cantidad a pagar.

Características generales IX

- La función de selección elige la moneda de mayor valor que queda en el conjunto de candidatos.

Características generales IX

- La función objetivo cuenta el número de monedas utilizadas en la solución.

Características generales X

- Obviamente, es más eficiente rechazar todas las monedas de 100 unidades restantes (digamos) de una vez cuando la cantidad restante a representar cae por debajo de este valor.

Características generales X

- Usar la división de enteros para calcular cuántas monedas de un valor particular se deben elegir, también es más eficiente que proceder mediante sustracciones sucesivas.

Características generales X

- Si se adopta alguna de estas tácticas, podemos relajar la condición de que el conjunto de monedas disponible debe ser finito.

Árboles de expansión mínimos I

Sea $G = \langle N, A \rangle$ un grafo no dirigido y conectado donde N es el conjunto de nodos y A es el conjunto de aristas. Cada arista tiene una longitud no negativa dada.

Árboles de expansión mínimos I

Sea $G = \langle N, A \rangle$ un grafo no dirigido y conectado donde N es el conjunto de nodos y A es el conjunto de aristas. Cada arista tiene una longitud no negativa dada.

El problema es encontrar un subconjunto T de las aristas de G tal que todos los nodos permanezcan conectados cuando sólo se usan las aristas en T y la suma de las longitudes de las aristas en T es lo más pequeña posible.

Árboles de expansión mínimos II

- Dado que G está conectado, debe existir al menos una solución.

Árboles de expansión mínimos II

- Si G tiene aristas de longitud 0, entonces pueden existir varias soluciones cuya longitud total sea la misma pero que involucren diferentes números de aristas.

Árboles de expansión mínimos II

- En este caso, dadas dos soluciones con la misma longitud total, preferimos la que tiene menos aristas.

Árboles de expansión mínimos II

- Incluso con esta salvedad, el problema puede tener varias soluciones diferentes de igual valor.

Árboles de expansión mínimos II

- En lugar de hablar de longitud, podemos asociar un costo a cada arista.

Árboles de expansión mínimos II

- El problema es entonces encontrar un subconjunto T de las aristas cuyo costo total sea lo más pequeño posible.

Árboles de expansión mínimos II

- Evidentemente, este cambio de terminología no afecta la forma en que resolvemos el problema.

Árboles de expansión mínimos III

- Sea $G' = \langle N, T \rangle$ el grafo parcial formado por los nodos de G y las aristas en T y suponga que hay n nodos en N.

Árboles de expansión mínimos III

- Un grafo conectado con n nodos debe tener al menos $n-1$ aristas, por lo que este es el número mínimo de aristas que puede haber en T.

Árboles de expansión mínimos III

- Por otro lado, un grafo con n nodos y más de $n-1$ aristas contiene al menos un ciclo; vea el problema 6.7 del libro de texto.

Árboles de expansión mínimos III

- Por tanto, si G' está conectado y T tiene más de $n-1$ aristas, podemos eliminar al menos una de estas sin desconectar G', siempre que elijamos una arista que sea parte de un ciclo.

Árboles de expansión mínimos IV

- Esto disminuirá la longitud total de las aristas en T o dejará la longitud total igual (si hemos eliminado una arista con longitud 0) mientras que disminuye el número de aristas en T.

Árboles de expansión mínimos IV

- En cualquier caso, la nueva solución es preferible a la anterior.

Árboles de expansión mínimos IV

- Por tanto, un conjunto T con n o más aristas no puede ser óptimo. De ello se deduce que T debe tener exactamente $n-1$ aristas y dado que G' está conectado, debe ser un árbol.

Árboles de expansión mínimos IV

- El grafo G' se llama árbol de expansión mínimo para el grafo G.

Árboles de expansión mínimos IV

- Este problema tiene muchas aplicaciones.

Árboles de expansión mínimos IV

- Por ejemplo, suponga que los nodos de G representan ciudades y que el costo de una arista $\{a, b\}$ sea el costo de colocar una línea telefónica de a a b.

Árboles de expansión mínimos V

- Entonces, un árbol de expansión mínimo de G corresponde a la red más barata posible que sirve a todas las ciudades en cuestión, siempre que sólo se puedan usar enlaces directos entre ciudades (en otras palabras, siempre que no se nos permita construir centrales telefónicas en el campo entre las ciudades).

Árboles de expansión mínimos V

- Relajar esta condición equivale a permitir la adición de nodos auxiliares adicionales a G. Esto puede permitir que se obtengan soluciones más baratas: consulte el problema 6.8 del libro de texto.

Árboles de expansión mínimos V

- A primera vista, al menos dos líneas de ataque parecen posibles si esperamos encontrar un algoritmo voraz para este problema.

Árboles de expansión mínimos V

- Claramente, nuestro conjunto de candidatos debe ser el conjunto A de aristas en G.

Árboles de expansión mínimos VI

- Una posible táctica es comenzar con un conjunto T vacío y seleccionar en cada etapa la arista más corta que aún no se ha elegido o rechazado, independientemente de dónde se encuentre situada esta arista en G.

Árboles de expansión mínimos VI

- Otra línea de ataque implica elegir un nodo y construir un árbol desde allí, seleccionando en cada etapa la arista más corta disponible que pueda extender el árbol a un nodo adicional.

Árboles de expansión mínimos VI

- ¡Inusualmente, para este problema particular ambos enfoques funcionan!

Árboles de expansión mínimos VI

- Antes de presentar los algoritmos, mostramos cómo se aplica el esquema general de un algoritmo voraz en este caso y presentamos un lema para su uso posterior.

Árboles de expansión mínimos VII

- Los candidatos, como ya se señaló, son las aristas en G.

Árboles de expansión mínimos VII

- Un conjunto de aristas es una solución si constituye un árbol de expansión para los nodos en N.

Árboles de expansión mínimos VII

- Un conjunto de aristas es factible si no incluye un ciclo.

Árboles de expansión mínimos VII

- La función de selección que usamos varía con el algoritmo.

Árboles de expansión mínimos VII

- La función objetivo a minimizar es la longitud total de las aristas en la solución.

Árboles de expansión mínimos VIII

- También necesitamos más terminología. Decimos que un conjunto factible de aristas es prometedor si puede extenderse para producir no solo una solución, sino una solución óptima a nuestro problema. En particular, el conjunto vacío siempre es prometedor (ya que siempre existe una solución óptima).

Árboles de expansión mínimos VIII

- Además, si un conjunto de aristas prometedor ya es una solución, entonces la extensión requerida es vacía y esta solución debe ser óptima.

Árboles de expansión mínimos VIII

- A continuación, decimos que una arista abandona un conjunto dado de nodos si exactamente un extremo de esta arista está en el conjunto.

Árboles de expansión mínimos VIII

- Por lo tanto, una arista no puede dejar un conjunto dado de nodos porque ninguno de sus extremos está en el conjunto o, menos evidentemente, porque ambos lo están.

Árboles de expansión mínimos VIII

- El siguiente lema es crucial para demostrar la corrección de los próximos algoritmos.

Árboles de expansión mínimos IX

Lema 1

Sea $\langle N, A \rangle$ un grafo conectado no dirigido donde cada arista tiene asociada una longitud. Sea $B \subset N$ un subconjunto propio de los nodos de G. Sea $T \subseteq A$ un conjunto de aristas prometedor tal que ninguna arista en T deja a B. Sea v la arista más corta que deja B (o una de las más cortas si hay empate). Entonces $T \cup \{v\}$ es prometedor.

Árboles de expansión mínimos X

Demostración: Sea U un árbol de expansión mínimo de G tal que $T \subseteq U$. Tal U debe existir ya que T es prometedor por suposición. Si $v \in U$, no hay nada que probar.

Árboles de expansión mínimos X

Demostración: Sea U un árbol de expansión mínimo de G tal que $T \subseteq U$. Tal U debe existir ya que T es prometedor por suposición. Si $v \in U$, no hay nada que probar.

De lo contrario, cuando agregamos la arista v a U, creamos exactamente un ciclo (ésta es una de las propiedades de un árbol).

Árboles de expansión mínimos X

Demostración: Sea U un árbol de expansión mínimo de G tal que $T \subseteq U$. Tal U debe existir ya que T es prometedor por suposición. Si $v \in U$, no hay nada que probar.

De lo contrario, cuando agregamos la arista v a U, creamos exactamente un ciclo (ésta es una de las propiedades de un árbol).

En este ciclo, dado que v sale de B, necesariamente existe al menos otra arista, digamos u, que también sale de B, o el ciclo no podría cerrarse.

Demostración: Sea U un árbol de expansión mínimo de G tal que $T \subseteq U$. Tal U debe existir ya que T es prometedor por suposición. Si $v \in U$, no hay nada que probar.

De lo contrario, cuando agregamos la arista v a U, creamos exactamente un ciclo (ésta es una de las propiedades de un árbol).

En este ciclo, dado que v sale de B, necesariamente existe al menos otra arista, digamos u, que también sale de B, o el ciclo no podría cerrarse.

Si ahora quitamos u, el ciclo desaparece y obtenemos un nuevo árbol V que expande a G.

Árboles de expansión mínimos XI

Sin embargo, la longitud de v por definición no es mayor que la longitud de u, y por lo tanto la longitud total de las aristas en V no excede la longitud total de las aristas en U.

Árboles de expansión mínimos XI

Sin embargo, la longitud de v por definición no es mayor que la longitud de u, y por lo tanto la longitud total de las aristas en V no excede la longitud total de las aristas en U.

Por lo tanto, V es también un árbol de expansión mínimo de G, e incluye v.

Árboles de expansión mínimos XI

Sin embargo, la longitud de v por definición no es mayor que la longitud de u, y por lo tanto la longitud total de las aristas en V no excede la longitud total de las aristas en U.

Por lo tanto, V es también un árbol de expansión mínimo de G, e incluye v.

Para completar la demostración, queda señalar que $T \subseteq V$ porque la arista u que se eliminó deja B, y por lo tanto no podría haber sido una arista de T.

Algoritmo de Kruskal I

- El conjunto T de aristas está inicialmente vacío.

Algoritmo de Kruskal I

- A medida que avanza el algoritmo, se agregan aristas a T.

Algoritmo de Kruskal I

- Siempre que no haya encontrado una solución, el grafo parcial formado por los nodos de G y las aristas en T consta de varios componentes conectados (inicialmente, cuando T está vacío, cada nodo de G forma un componente conectado trivial distinto).

Algoritmo de Kruskal I

- Los elementos de T incluidos en un componente conectado dado forman un árbol de expansión mínimo para los nodos de este componente.

Algoritmo de Kruskal I

- Al final del algoritmo, solo queda un componente conectado, por lo que T es un árbol de expansión mínimo para todos los nodos de G.

Algoritmo de Kruskal II

- Para construir componentes conectados cada vez más grandes, examinamos las aristas de G en orden de longitud creciente.

Algoritmo de Kruskal II

- Si una arista une dos nodos en diferentes componentes conectados, lo agregamos a T. En consecuencia, los dos componentes conectados ahora forman un solo componente.

Algoritmo de Kruskal II

- De lo contrario, la arista se rechaza: ya que une dos nodos en el mismo componente conectado y, por lo tanto, no se puede agregar a T sin formar un ciclo (porque las aristas en T forman un árbol para cada componente).

Algoritmo de Kruskal II

- El algoritmo se detiene cuando sólo queda un componente conectado.

Algoritmo de Kruskal III

Para ilustrar cómo funciona este algoritmo, considere el grafo de la Figura 1.

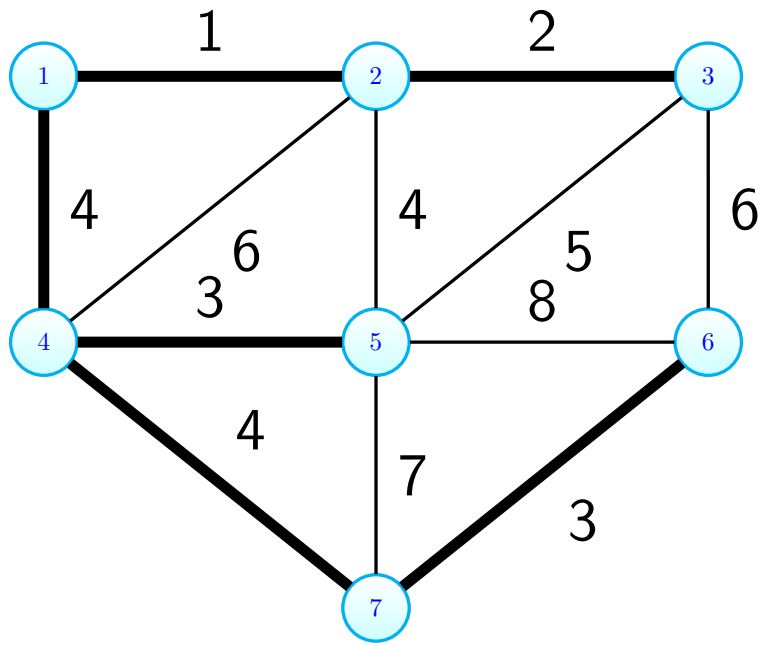

Figure 1: Un grafo y su árbol de expansión mínimo

En orden creciente de longitud, las aristas son: $\{1,2\}, \{2,3\}, \{4,5\}, \{6,7\}, \{1,4\}, \{2,5\}, \{4,7\}, \{3,5\}, \{2,4\}, \{3,6\}, \{5,7\}$ y $\{5,6\}$.

Algoritmo de Kruskal IV

El algoritmo procede como sigue:

Paso	Arista considerada	Componentes conectadas
Inicio		$\{1\}\{2\}\{3\}\{4\}\{5\}\{6\}\{7\}$
1	$\{1, 2\}$	$\{1, 2\}$ $\{3\}$ $\{4\}$ $\{5\}$ $\{6\}$ $\{7\}$
2	$\{2, 3\}$	$\{1, 2, 3\}$ $\{4\}$ $\{5\}$ $\{6\}$ $\{7\}$
3	$\{4, 5\}$	$\{1, 2, 3\}$ $\{4, 5\}$ $\{6\}$ $\{7\}$
4	$\{6, 7\}$	$\{1, 2, 3\}$ $\{4, 5\}$ $\{6, 7\}$
5	$\{1, 4\}$	$\{1, 2, 3$,4 , $5\}$ $\{6, 7\}$
6	$\{2, 5\}$	rechazado
7	$\{4, 7\}$	$\{1, 2, 3, 4, 5, 6, 7\}$

Algoritmo de Kruskal IV

El algoritmo procede como sigue:

Paso	Arista considerada	Componentes conectadas
Inicio		$\{1\}\{2\}\{3\}\{4\}\{5\}\{6\}\{7\}$
1	$\{1, 2\}$	$\{1, 2\}$ $\{3\}$ $\{4\}$ $\{5\}$ $\{6\}$ $\{7\}$
2	$\{2, 3\}$	$\{1, 2, 3\}$ $\{4\}$ $\{5\}$ $\{6\}$ $\{7\}$
3	$\{4, 5\}$	$\{1, 2, 3\}$ $\{4, 5\}$ $\{6\}$ $\{7\}$
4	$\{6, 7\}$	$\{1, 2, 3\}$ $\{4, 5\}$ $\{6, 7\}$
5	$\{1, 4\}$	$\{1, 2, 3$,4 , $5\}$ $\{6, 7\}$
6	$\{2, 5\}$	rechazado
7	$\{4, 7\}$	$\{1, 2, 3, 4, 5, 6, 7\}$

Cuando el algoritmo se detiene, T contiene las aristas elegidas $\{1, 2\}, \{2, 3\}$, $\{4, 5\}, \{6, 7\}, \{1, 4\}$ y $\{4, 7\}$. Este árbol de expansión mínimo se muestra con líneas gruesas en la Figura 1; su longitud total es 17.

Algoritmo de Kruskal V

Teorema 2

El algoritmo de Kruskal encuentra un árbol de expansión mínimo.

Algoritmo de Kruskal V

Teorema 2

El algoritmo de Kruskal encuentra un árbol de expansión mínimo.

Demostración:

- La prueba es por inducción matemática sobre el número de aristas en el conjunto T.

Algoritmo de Kruskal V

Teorema 2

El algoritmo de Kruskal encuentra un árbol de expansión mínimo.

Demostración:

- La prueba es por inducción matemática sobre el número de aristas en el conjunto T.
- Demostraremos que si T es prometedor en cualquier etapa del algoritmo, entonces sigue siendo prometedor cuando se ha agregado una arista extra.

Algoritmo de Kruskal V

Teorema 2

El algoritmo de Kruskal encuentra un árbol de expansión mínimo.

Demostración:

- La prueba es por inducción matemática sobre el número de aristas en el conjunto T.
- Demostraremos que si T es prometedor en cualquier etapa del algoritmo, entonces sigue siendo prometedor cuando se ha agregado una arista extra.
- Cuando el algoritmo se detiene, T da una solución a nuestro problema; como también es prometedora, esta solución es óptima.

Algoritmo de Kruskal VI

Base: El conjunto vacío es prometedor porque G está conectado y, por lo tanto, debe existir una solución.

Algoritmo de Kruskal VI

Paso inductivo: Suponga que T es prometedor justo antes de que el algoritmo agregue una nueva arista $e = \{u, v\}$.

Algoritmo de Kruskal VI

Las aristas en T dividen los nodos de G en dos o más componentes conectados; el nodo u está en uno de estos componentes y v está en un componente diferente.

Algoritmo de Kruskal VI

Sea B el conjunto de nodos del componente que incluye a u. Ahora

Algoritmo de Kruskal VII

- el conjunto B es un subconjunto propio de los nodos de G (ya que no incluye v, por ejemplo);

Algoritmo de Kruskal VII

- T es un conjunto prometedor de aristas de manera que ninguna arista en T sale de B (ya que una arista en T tiene ambos extremos en B o no tiene ningún extremo en B, por lo que por definición no sale de B); y

Algoritmo de Kruskal VII

- e es una de las aristas más cortas que sale de B (ya que todas las aristas estrictamente más cortas ya se han examinado, y se han incorporado a T o se han rechazado porque tenían ambos extremos en el mismo componente conectado).

Algoritmo de Kruskal VII

Por tanto, se cumplen las condiciones del Lema 1 y concluimos que el conjunto $T \cup \{e\}$ también es prometedor.

Algoritmo de Kruskal VII

Esto completa la demostración por inducción matemática de que el conjunto T es prometedor en cada etapa del algoritmo y, por lo tanto, cuando el algoritmo se detiene, T no sólo da una solución a nuestro problema, sino una solución óptima.

Algoritmo de Kruskal VIII

- Para implementar el algoritmo, tenemos que manejar un cierto número de conjuntos, es decir, los nodos en cada componente conectado.

Algoritmo de Kruskal VIII

- Hay que realizar dos operaciones rápidamente: $\textsc{find}(x)$, que nos dice en qué componente conectado se encuentra el nodo x, y $\textsc{merge}(A, B)$, para fusionar dos componentes conectados.

Algoritmo de Kruskal VIII

- Por lo tanto, utilizamos estructuras de conjuntos disjuntos; consulte la Sección 5.9 del libro de texto.

Algoritmo de Kruskal VIII

- Para este algoritmo es preferible representar el grafo como un vector de aristas con sus longitudes asociadas en lugar de como una matriz de distancias; vea el problema 6.9 del libro de texto.

Algoritmo de Kruskal IX

Aquí está el algoritmo.

function $\textsc{Kruskal}(G = \langle N, A\rangle : graph, lenght : A \to \mathbb{R}^+)$

1 $A \leftarrow \textsc{sortByIncreasingLength}(A)$
2 $n \leftarrow \textsc{numberOfNodesIn}(N)$
3 $T \leftarrow \emptyset$
4 Inicializar n conjuntos, cada uno con un elemento diferente de N
5 **repeat**
6 $\quad e \leftarrow \{u, v\}$ la arista más corta aún no considerada
7 $\quad ucomp \leftarrow \textsc{find}(u)$
8 $\quad vcomp \leftarrow \textsc{find}(v)$
9 $\quad$ **if** $ucomp \neq vcomp$
10 $\quad\quad$ **then** $\textsc{merge}(ucomp, vcomp)$
11 $\quad\quad\quad T \leftarrow T \cup \{e\}$
12 **until** T contenga $n - 1$ aristas
13 **return** T

Algoritmo de Kruskal X

Podemos evaluar el tiempo de ejecución del algoritmo de la siguiente manera. En un grafo con n nodos y a aristas, el número de operaciones está en

- $\Theta(a \log a)$ para ordenar las aristas, lo cual es equivalente a $\Theta(a \log n)$ ya que $n - 1 \leq a \leq n(n-1)/2$;

Algoritmo de Kruskal X

Podemos evaluar el tiempo de ejecución del algoritmo de la siguiente manera. En un grafo con n nodos y a aristas, el número de operaciones está en

- $\Theta(n)$ para inicializar los n conjuntos disjuntos;

Algoritmo de Kruskal X

Podemos evaluar el tiempo de ejecución del algoritmo de la siguiente manera. En un grafo con n nodos y a aristas, el número de operaciones está en

- $\Theta(2a\alpha(2a, n))$ para todas las operaciones FIND y MERGE, donde α es la función de crecimiento lento definida en la Sección 5.9 del libro de texto (esto se deduce de los resultados de la Sección 5.9 del libro de texto, ya que hay como máximo $2a$ operaciones de búsqueda y $n - 1$ operaciones de fusión en un universo que contiene n elementos); y

Algoritmo de Kruskal X

Podemos evaluar el tiempo de ejecución del algoritmo de la siguiente manera. En un grafo con n nodos y a aristas, el número de operaciones está en

- en el peor de los casos $O(a)$ para las operaciones restantes.

Algoritmo de Kruskal XI

- Concluimos que el tiempo total del algoritmo está en $\Theta(a \log n)$ porque $\Theta(\alpha(2a, n)) \subset \Theta(\log n)$.

Algoritmo de Kruskal XI

- Concluimos que el tiempo total del algoritmo está en $\Theta(a \log n)$ porque $\Theta(\alpha(2a, n)) \subset \Theta(\log n)$.
- Aunque esto no cambia el análisis del peor caso, es preferible mantener las aristas en un heap invertido (consulte la Sección 5.7 del libro de texto): por lo tanto, la arista más corta está en la raíz del heap.

Algoritmo de Kruskal XI

- Concluimos que el tiempo total del algoritmo está en $\Theta(a \log n)$ porque $\Theta(\alpha(2a, n)) \subset \Theta(\log n)$.
- Aunque esto no cambia el análisis del peor caso, es preferible mantener las aristas en un heap invertido (consulte la Sección 5.7 del libro de texto): por lo tanto, la arista más corta está en la raíz del heap.
- Esto permite que la inicialización se realice en un tiempo en $\Theta(a)$, aunque cada búsqueda de un mínimo en el ciclo de repetición ahora lleva un tiempo en $\Theta(\log a) = \Theta(\log n)$.

Algoritmo de Kruskal XI

- Concluimos que el tiempo total del algoritmo está en $\Theta(a \log n)$ porque $\Theta(\alpha(2a, n)) \subset \Theta(\log n)$.
- Aunque esto no cambia el análisis del peor caso, es preferible mantener las aristas en un heap invertido (consulte la Sección 5.7 del libro de texto): por lo tanto, la arista más corta está en la raíz del heap.
- Esto permite que la inicialización se realice en un tiempo en $\Theta(a)$, aunque cada búsqueda de un mínimo en el ciclo de repetición ahora lleva un tiempo en $\Theta(\log a) = \Theta(\log n)$.
- Esto es particularmente ventajoso si el árbol de expansión mínimo se encuentra en un momento en el que quedan por probar un número considerable de aristas.

Algoritmo de Kruskal XI

- Concluimos que el tiempo total del algoritmo está en $\Theta(a \log n)$ porque $\Theta(\alpha(2a, n)) \subset \Theta(\log n)$.
- Aunque esto no cambia el análisis del peor caso, es preferible mantener las aristas en un heap invertido (consulte la Sección 5.7 del libro de texto): por lo tanto, la arista más corta está en la raíz del heap.
- Esto permite que la inicialización se realice en un tiempo en $\Theta(a)$, aunque cada búsqueda de un mínimo en el ciclo de repetición ahora lleva un tiempo en $\Theta(\log a) = \Theta(\log n)$.
- Esto es particularmente ventajoso si el árbol de expansión mínimo se encuentra en un momento en el que quedan por probar un número considerable de aristas.
- En tales casos, el algoritmo original pierde tiempo ordenando estas aristas inútiles.

Ejercicios

- Equipo 1: Algoritmo de Prim, sección 6.3.2 del libro de texto.
- Equipo 2: Algoritmo de Dijkstra, sección 6.4 del libro de texto.
- Equipo 3: Algoritmo de la mochila, sección 6.5 del libro de texto.
- Equipo 4: Algoritmo de planificación, sección 6.6.1 del libro de texto.
- Equipo 5: Algoritmo de planificación con límites de tiempo, sección 6.6.2 del libro de texto.
- Equipo 6: Algoritmo de planificación con límites de tiempo, sección 6.6.2 del libro de texto.

Objetivos

Desarrollar habilidades para:

Objetivos

Desarrollar habilidades para:

- recopilar información,

Objetivos

Desarrollar habilidades para:

- organizar información,

Objetivos

Desarrollar habilidades para:

- asimilar conocimiento nuevo,

Objetivos

Desarrollar habilidades para:

- transmitir conocimientos,

Objetivos

Desarrollar habilidades para:

- estructurar adecuadamente una exposición,

Objetivos

Desarrollar habilidades para:

- argumentar adecuadamente una exposición.

Objetivos

Desarrollar habilidades para:

- recopilar información,
- organizar información,
- asimilar conocimiento nuevo,
- transmitir conocimientos,
- estructurar adecuadamente una exposición,
- argumentar adecuadamente una exposición.

Reglas

1. Cada equipo tiene 15 minutos para exponer y 5 minutos para preguntas.

Reglas

2. Todos los miembros de cada equipo deben exponer.

Reglas

3. Enunciar claramente el problema a resolver y la técnica de diseño del algoritmo que lo resuelve.

Reglas

4. Ejemplificar el funcionamiento del algoritmo.

Reglas

5. Demostrar que el algoritmo es correcto.

Reglas

6. Presentar el algoritmo.

Reglas

7. Analizar la eficiencia del algoritmo.

Reglas

8. No se pueden hacer intervenciones mientras un equipo está exponiendo.

Reglas

9. Si alguien quiere preguntar debe anotar el número de diapositiva donde le surgió la duda.

Reglas

10. Si no terminan en 15 minutos la exposición o si no terminan de responder la(s) pregunta(s) en los cinco minutos, serán penalizados con un punto en la calificación.

Reglas

11. No se pueden extender, una vez terminado el tiempo se les quitará la palabra.

Reglas

12. La presentación se debe hacer en LaTeX, usando la clase beamer.

Reglas

13. Si tienen que incluir alguna imagen, la pueden construir fuera de LaTeX y luego importarla a su documento.

Reglas

14. Los puntos 4, 5 y 6 pueden variar dependiendo de las preferencias de cada equipo.

Reglas

1. Cada equipo tiene 15 minutos para exponer y 5 minutos para preguntas.
2. Todos los miembros de cada equipo deben exponer.
3. Enunciar claramente el problema a resolver y la técnica de diseño del algoritmo que lo resuelve.
4. Ejemplificar el funcionamiento del algoritmo.
5. Demostrar que el algoritmo es correcto.
6. Presentar el algoritmo.
7. Analizar la eficiencia del algoritmo.
8. No se pueden hacer intervenciones mientras un equipo está exponiendo.
9. Si alguien quiere preguntar debe anotar el número de diapositiva donde le surgió la duda.
10. Si no terminan en 15 minutos la exposición o si no terminan de responder la(s) pregunta(s) en los cinco minutos, serán penalizados con un punto en la calificación.
11. No se pueden extender, una vez terminado el tiempo se les quitará la palabra.
12. La presentación se debe hacer en LaTeX, usando la clase beamer.
13. Si tienen que incluir alguna imagen, la pueden construir fuera de LaTeX y luego importarla a su documento.
14. Los puntos 4, 5 y 6 pueden variar dependiendo de las preferencias de cada equipo.