TC2017_NP1_A01020200
Actividad no presencial 1
Juan Carlos León

Complexity
P- tiempo polinomial -> set de problemas que peudes resolver en tiempo polinomial
Exp- tiempo exponencial -> set de todos los problemas que peudes resolver en tiempo exponencial

R- tiempo finito->todos los problemas que se pueden resolver en un tiempo determinado->es decir un problema computable
Pero hay problemas que son mas complejos que estos

Problemas de decisión son en los que la respuesta
es SI o NO
NP- set de problemas de deciciones-se parece a P
NP- nondeterministic polynomial

NP- todos los problemas que se pueden resolver en tiempo polinomial pero con un algoritmo "suertudo"
NP- problemas que tiene soluciones que peuden revisarse en tiempo polinomial

Reducccion- tiene un problema lo haces un grafo y aplicas algoritmos de grafo y se resuelve
Convertir un problema A que quieres resolver en un problema B que ya sabes resolver
Para resolver problemas problemas reciclando código

Divide y Venceras
Es separar en pedazos mas pequeños un problema en el que los peudas resolver de manera individual.
Técnica de diseño de algoritmos.

1. Divide
   1.1. Dividir el problema en sub problemas
   1.2. Deben ser mas pequeños (el valor de n sea menor)
2. Venceras
   2.1. Resolver cada subproblema recursivamente
3. Combina
   3.1. Combinar las pequeñas soluciones en una solución para el problema

EJ: merge sort
Binary search – solo se recursea en un lado del arreglo

Como representar grafos
Matriz de adyacencia- requiere almacenamiento n^2 -> no es buena cuando hay muchos 0
Lista de adyacencia- el set a los que ese nodo tiene coneccion -> almacenamiento - usa v*e espacio en memoria
Hand shaking lemma – si para un grafo no direccionado sumo los degrees de cada vértice sera el doble de los edges
2*|e|
Mínimum spanning - muy importante distributed system <- lo buscan
Spanning tree conecta todo slos vertices con weight minimo
Hace la eleccion mas optima esperando que el optimo global se alcance
Problemas al combinar

Lecture 19: Dynamic Programming I: Fibonacci, Shortest Paths
La programacion dinamica es general y poderosa, algunas formas para emplearla es para encontrar los caminos mas cortos y la optimizacion. Es como un “carefull brute forcé”, probar las posibilidades cuidadosamente, esto para obtener un tiempo polinomial.
La programación dinámica (DP,por sus siglas en ingles) la invento Richard bellman. Lo llamo DP para “ocultar” que estaba haciendo investigación matemática.
DP ~ tener un problema -> dividirlo en subproblemitas y reuar la solucion

La primera vez que es la non-memoized hace una recursividad y para los siguientes solo llamas al memoized. Las llamadas memoized tienen un costo de tiempo constante. Por esto podemos decir que la recucion inicial + las llamadas memoized nos dan un tiempo lineal. Θ(n).

Solo contamos cada subproblema una vez que es en la que trabajo y no por cada recurcion porque como ya lo tengo memoizado seria contarlo doble.

Caminos mas cortos
Adivinar: cuando no sabes la solución adivinas, y pruebas todas las adivinadas. Y tomar la mejor
DP = recursividad + memoizar + adivinar.

Para ver el camino mas corto vemos desde el final hacia el principio. Los nodos que llegan a V que a dodne quiero llegar y luego uego los que llegan a ese y asi hasta llegar al de inicio.

Lecture 20: Dynamic Programming II: Text Justification, Blackjack

DAG: directed acyclic graph. Grafica direccionada sin ciclos

Time = # sub problems * time/sub problem treating recursive calls as θ(1)

5 pasos generals: (no osn nesesariamente sequenciales/son medio independientes)

- Define sub problems
  - # de sub problemas
- Adivinar (parte de la solución)
  - # de opciones para adivinar (cuantas posibilidades)
- Relacionar la solución de sub problemas
  - Normalmente con recursividad
  - Tiempo / sub problema
    - § Que sea polinomial
    - § Similar al # de opciones
- Construir el algoritmo
  - Recursivo y memoizado
  - Bottom up approach (building a table)
  - Que la recursion de sub problema sea aciclica
    - § Orden topologico
    - § Para poder usar ordenamiento topologico
  - El orden en que vas a hacer las cosas
    - § Ej: Fibonacci empezar con los mas chicos
  - Tiempo total = # de sub problemas * tiempo / sub problema
- Resolver el problea original
  - Asegurarse que el problema que querias resolver en verdad se resuelva
  - Aveses requiere un poco de tiempo extra juntar todas las soluciones

Justificacion de textos

Definen un rango de “badness”. Que tan malo es usar las palabras de i a j-1 en esa línea. Pueden o no caber. Si la suma de caracteres y espacios en la línea es mayor que lo que cabe, entonces no cabe. Si no cabe se define “badness” como infinito. Si cabe “badness” va a ser ((longitud máxima)-(longitud total de la línea))^3 . si están cerca entonces va a ser 0 que es bueno. Si la línea es muy corta al ^3 el “badness” sera mucho mas grande. Lo que se intenta es minimizar la suma de todos los “badness” de todas las líneas.

Hay que adivinar después de la primera palabra donde empieza la segunda linea, después donde la 3, etc. así tengo que mis sub problemas son como el original. Los sub problemas en este caso serán el sufijo (lo que queda del arreglo) después de que corte para hacer un nuevo renglón.

Parent pointers: acordarse cual de las adivinadas fue la mejor. En ete caso es el mejor valor que hizo que la partición fuera correcta. 0 que es la primera línea. 0-> parent[0]->parent[parent[0]].

Lecture 21: Dynamic Programming III: Parenthesization, Edit Distance, Knapsack

Como elegir sub problemas:

- Para strings y secuencias
    - Aveces Usamos sufijos
    - Eso nos deja con n o n+1 sub problemas
    - Aveces usamos prefijos (como un sufijo pero la parte que quitamos es del final y no del principio)
    - También es lienal
    - Sub strings
        - § Tienen que ser consecutivas
        - § De i a j por ejemplo
    - Son esta tenemos n^2

Parenthesization

Te dan una exprecion asosiativa y quieres evaluarla en algún orden

Adivinamos cual seria la ultima operación que vamos a hacer. (A0….Ak-1)*(Ak…An-1) y esa seria la ultima multiplicación.

Todas las posibilidades son k. y buscamos la optima k para llegar a la formula. (A0….Ak-1)esto es un prefijo.

(Ak…An-1) esto es un sufijo.

Pero al hacer el paso recusivo en este, te vas a ver prácticamente obligado a usar substring también.