# Programación Dinámica

# Definición

- Filofofía de Divide & Conquer

- Dos características que debe tener el problema:
    1) Resursividad
    2) Sub-optimalidad (?)

# Filosofía de la programación dinámica

¿Cuándo se piensa en programación dinámica?

Dado un conjunto de instancias de un mismo problema sobre diferentes estados o configuraciones (nodos), cada una dependiendo de algunas otras (aristas). La programación dinámica es aplicable cuando el grafo formado es un DAG.

# Tipos de dinámica

1. Clásica:

   Procesamiento de las instancias en orden

2. Memoized

   Modo recursivo estandard, sin recálculo de instancias

# Ejemplos

- Fibonacci
- Números factoriales
- Longest increasing subsequence
- Longest common subsequence
- Dijkstra, (Pink-)Floyd-Warshall, etc.

# Pink-Floyd-Warshall

Problema: Dado un grafo con peso (no negativo), calcular la mínima distancia de todos los nodos a todos los nodos.

Solución: Dinámica, condicionando en el subconjunto de elementos que se pueden usar en medio del camino.

# Mochilas

- Dados n objetos con pesos y valores, y una mochila con cierta capacidad, llenarla con el mayor valor posible.

  (http://uva.onlinejudge.org/external/5/562.html)

# Máscaras de bits

Las máscaras de bits permiten manejar información de conjuntos de pocos elementos, son muy usuales en la programación dinámica.

- Hamiltonian path-cycle
- dependencia sobre subconjuntos varios
- Long night of museums! (http://livearchive.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=14&page=show_problem&problem=1154)

# Máscaras de bits

Un número puede ser visto como un conjunto ordenado de bits, y esto puede ser pensado como una representación de un conjunto, donde 1 indica que un elemento está en el conjunto y 0 indica lo contrario.

Sugerencia: http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=bitManipulation

Operaciones con bits:

- Unión: Or
- Intersección: And
- Diferencia simétrica: Xor
- Complemento: Not
- Último bit: x & (x-1)

...

- Recorrer todos los subconjuntos del conjunto s:

```
for ( int mask = s ; mask != 0 ; mask = (mask - 1) & s )
```

Problema ejemplo: Un conjunto se dice de suma prima si la suma de sus elementos es un número primo. Definimos la primalidad de un conjunto A como el máximo k tal que existen A_1; A_2; ... ; A_k = A tales que son todos de suma prima con A_i incluido en A_{i+1}, o 0 si A no es de suma prima. Dados n números enteros positivos distintos (1 <= n <= 10) calcular la primalidad del conjunto compuesto por esos n números

...

Acá el código fuente:

```
int mask[1024];
int conjunto[10];
bool esDeSumaPrima(int n){
    int t = 0;
    for(int i=0;i<10;i++){
        if((n&(1<<i)!=0) t += conjunto[i];
    }
    if(esPrimo(t)) return true;
    else return false;
}
```

...

```
int primalidad(int n)
{
    if(!esDeSumaPrima(n))
        return 0;
    if(mask[n]!=-1)
        return mask[n];
    mask[n] = 1;
    for(int i=n; i!=0; i = (i-1)&n){
      mask[n] = max(mask[n],primalidad(i)+1);
    }
return mask[n];
}
```

...

Ejemplo: Looking for order

# Dinámica con maps (para C++)

En situaciones en que guardar todas las instancias posibles es inimplementable o dificil, puede recurrirse a los maps de la STL. Como por ejemplo si los estados son permutaciones, o secuencias ordenadas.

Ejemplo: Dado un grafo de n nodos (n <= 20), inicialmente sin aristas, en cada turno se agrega una arista al azar entre dos nodos (pueden repetirse aristas). Calcular la esperanza del tiempo transcurrido hasta que el grafo sea conexo.

estados: {a_1, ..., a_k} donde k es la cantidad de componentes conexas, a_i la cantidad de nodos de cada componente.

# Moverse de a uno

Las dinámicas se aprenden con práctica, pero hay algunos trucos de implementación que se repiten y son útiles. Definitivamente el más importante de ellos es el "moverse de a uno", suele ahorrar un factor n en la complejidad algorítmica.

- Ejercicio: Mochila, ilimitada cantidad de cada uno de los objetos de peso w_1, ..., w_n y valor v_1, ..., v_n (v_i != v_j), mochila de tamaño C.

- Problema (**11766 - Racing Car Computer**): Hay n autos en una pista, un dispositivo en cada auto dice la cantidad de autos detrás y delante de ese auto. Algunos autos pueden estar lado a lado, los dispositivos no ven autos a la par.

...

Algunos dispositivos no funcionan correctamente. Determinar la mínima cantidad de dispositivos que están averiados.

Límites: n <= 1000; a,b<=1500

Solución: Dinámica! dp[k][many] = mínima cantidad de dispositivos averiados

Código principal:

...

```
int _dp(int k, int atras){
if(atras == n) return 0;
if(isc[k][atras]) return dp[k][atras];
if(k == si(car)) return n-atras;

int a = car[k].first.first, b = car[k].first.second, how = car[k].second;

int res = OO;
if(a == atras) res = (n-b-a) - how + _dp(k+1, n - b); //andan todos bien
if(a > atras) res = min(res, a - atras + (n-b-a) - how + _dp(k+1, n - b)); //hay algunos
defectuosos
res = min(res,  _dp(k+1, atras));

isc[k][atras] = true;
return dp[k][atras] = res;
}
```

...

- El truco de moverse de a uno se aplica cuando se busca un subconjunto de un conjunto con ciertas características, condicionando por ejemplo sobre si el primer elemento está o no.
- En caso de una dinámica donde se busque optimizar, no es necesario realizar la recursión sobre todos los nodos que prosiguen sino que basta ejecutar la recursión sobre los nodos adyacenes al estado actual.

# Ejemplos

Problema: (The islands, WF 2010)

Solución:

No tan dinámica dp[a][b][flag] = min(dp[a-1][b][flag'], dp[a][b-1][flag']); donde flag' se maneja adecuadamente.

dp[a][b][flag] = Solución optima dado que el primer camino arranca en el nodo a y el segundo en el nodo b

# Ejemplo

Problema: (Subway Timing, WF 2009)

Dado un arbol con peso en sus aristas, redondear los pesos a un múltiplo de 60 de modo de minimizar el máximo error en cualquier camino recorrido.

...

HINT 1: Los árboles suelen ser buenos candidatos para la estrategia Divide & Conquer. Muchas veces computar algo para un árbol con raíz R implica computarlo para cada subabol hijo de R y luego hacer una combinación de esos resultados.

HINT 2: Hay una cota bastante pequeña para el resultado. Busquenla.

HINT 3: La cota es 120. Sale laburando con los restos módulo 60 y el error en los caminos entre cada un nodo y la raíz.

...

HINT 4: No hay suboptimalidad en el problema. La solución óptima para un árbol no implica que se haya usado solución óptima para cada subárbol.
En estos casos, en lugar de intentar hallar la solución óptima, se suele encarar resolviendo el problema "Existe una solución <= X ?".

HINT 5: Para resolver si existe solución <= X con Divide & Conquer, busco solución <= X para cada uno de los subárboles y después me interesa saber el máximo error por abajo y máximo error por arriba a la raíz del subárbol.

...

HINT 6:  Cada subárbol puede tener varias soluciones <= X, de las cuales nos interesa saber
(A,B). A = máximo error por abajo, B = máximo error por arriba
Que tal si le pedimos que nos dé todos los posibles (A,B)?
Está bueno saber que A y B no son mayores que 120

Solución final: Dado un X (error permitible) y un nodo del arbol R, calculemos el conjunto {[a1,b1],....,[ak,bk]}, donde [ai,bi] representa una posible solución con error <= X del subárbol con raíz en R, siendo ai el error negativo mas grande y bi el error positivo mas grande entre un nodo del subárbol y R.

...

Notese que -120 <= ai <= bi <= 120 y que si tenemos ai <= aj <= bj <= bi, podemos eliminar el intervalo [ai,bi] porque no sirve para nada. Tambien es claro que ai <= 0 <= bi. Entonces la cantidad de intervalos que obtenemos como respuesta son a los sumo 121.

Ahora hay que combinar los resultados de los hijos en el padre. Para eso, se procesan los hijos uno por uno.
Sea la X las posibles soluciones que llevás para los primeros k hijos, e Y las soluciones para el hijo k+1-ésimo, y t el error que elegimos para la arista que une al padre con el hijo k+1-ésimo.

...

Para cada [a,b] en X y [c,d] en Y, probamos si se pueden combinar en una solución chequeando: a+(c+t) >= -X y b+(d+t) <= X
Si se puede, [min(a,c+t),max(b,d+t)] es una solución para los primeros k+1 hijos.

Si el conjunto de soluciones queda no vacío para la raíz del arbol original, es que era posible lograr el error X.
Después búsqueda binaria, o incluso búsqueda secuencial funciona.