Algoritmos para programación competitiva

Wilmer Emiro Castrillón Calderón

22 de octubre de 2021

Índice general

Prefacio

Este documento esta escrito con el objetivo de recopilar los algoritmos mas utilizados en programacióncompetitiva en un documento en español, exponer como funcionan, como utilizarlos y su implementación en C++,con explicaciones claras y precisas.

En este documento se asume que el lector ya tiene un conocimiento general del lenguaje C++, la librería estándarde C++, complejidades algorítmicas (notación O grande) y una experiencia por lo menos básica en competencias detipo ICPC.

Este documento contiene una lista de temas que personalmente trabaje a lo largo devarios años como competidor en maratones de ICPC y espero que sea de ayuda para los nuevos competidores.

Capítulo 1

Estructuras de datos

1.1. Tablas aditivas

Son estructuras de datos utilizadas para realizar operaciones acumuladas sobre un conjunto de datos estáticosen un rango específico, es decir, ejecutar una misma operación(como por ejemplo la suma) sobre un intervalode datos, se asume que los datos en la estructura no van a cambiar. Estas estructuras también son conocidas como*Summed-area table* para el procesamiento de imágenes, a pesar de su nombre no necesariamente sonexclusivas para operaciones de suma, pues la idea general es aplicable a otras operaciones.Durante el cálculo de una misma operación sobre diferentes rangos se presenta superposición de problemas, lastablas aditivas son utilizadas para reducir la complejidad computacional aprovechando estas superposicionesutilizando programación dinámica.

Ejemplo inicial.

Dado un vector V = {5,2,8,2,4,3,1} encontrar para múltiples consultas la suma de todos los elementos en un rango[i,j], indexando desde 1, por ejemplo con el rango [1, 3] la suma es [5+2+8] = 15.La solución trivial es hacer un ciclo recorriendo el vector entre el intervalo [i,j], en el peor de los casos sedebe recorrer todo el vector, esto tiene una complejidad $O(n)$ puede que para una consulta sea aceptable, peroen casos grandes como por ejemplo un vector de tamaño 10^5 y una cantidad igualmente grande de consultas eltiempo de ejecución se hace muy alto, por lo tanto se hace necesario encontrar una mejor solución.La operación suma tiene propiedades que nos pueden ayudar a resolver este problema de una forma mas eficiente:

1. La suma es asociativa es decir, se cumple: $a + (b + c) = (a + b) + c$, esto indica que sin importarla agrupación que se realice el resultado sera igual (no confundir con propiedad conmutativa).

2. La suma posee elemento neutro, es decir existe un β tal que $a + \beta = a$, en la suma$\beta = 0$.

3. La suma tiene operación inversa, es decir existe una operación que puede revertir la suma, la cuales la resta: si $a + b = c$ entonces $c - a = b$.

Considerando las anteriores propiedades el problema se puede trabajar desde otro enfoque, primero se puededefinir $suma(x) = \sum_{k=1}^{x} V_k$, ahora tomando como ejemplo una consulta en el rango [3,5]del vector V, se tomara convenientemente: $suma(2) = V_1 + V_2$ y $suma(5) = V_1 + V_2 + V_3 + V_4 + V_5$,pero se necesita encontrar $V_3 + V_4 + V_5$, usando la propiedad asociativa se obtiene:$(V_1 + V_2) + (V_3 + V_4 + V_5) = (V_1 + V_2 + V_3 + V_4 + V_5)$ y usando la propiedad inversa se llega a:$(V_3 + V_4 + V_5) = (V_1 + V_2 + V_3 + V_4 + V_5) - (V_1 + V_2)$ por lo tanto$(V_3 + V_4 + V_5) = suma(5) - suma(2)$. De esta manera el problema se puede generalizar como$\sum_{k=i}^{j} V_k = suma(j) - suma(i-1)$ cuando

$i \neq 1$ y $suma(j)$ cuando $i = 1$.Ahora pre-calculando $suma(x)$ se puede dar una solución inmediata a cada consulta, esto se puede resolverutilizando un enfoque básico de programación dinámica. Para encontrar $suma(x)$ se puede reescribir como:$suma(x) = V_x + suma(x-1)$ con caso base $suma(1) = V_1$ y por definición la operación acumulada sobre unconjunto vacío es igual al elemento neutro, a partir de esto se puede obtener la siguiente solución enC++ con consultas indexando desde 1.

```
int V[] = {5,2,8,2,4,3,1}, memo[8];

void precalcular(){
    memo[0] = 0;
    for(int i = 0; i < 7; i++)
        memo[i+1] = V[i] + memo[i];
}

int consulta(int i, int j){ return memo[j] - memo[i-1]; }
```

De manera general las tablas aditivas son aplicables a cualquier operación que posea las tres propiedadesdescritas anteriormente: ser asociativa, tener elemento neutro y operación inversa, por ejemplo la suma,multiplicación o el operador de bits XOR.

Tablas aditivas en 2D

Las operaciones acumuladas no solo se pueden usar sobre una dimension, sino también sobre n-dimensiones, en estoscasos se debe trabajar con el principio de inclusión-exclusión pues se debe considerar mejor las operaciones entreintervalos, si no tiene en cuenta este principio las soluciones contendrían elementos duplicados o faltantes, loque produciría soluciones incorrectas.**Ejemplo:** dada la matriz M encontrar para múltiples consultas la suma de todos los elemento en unasubmatriz $Q_{(i1,j1),(i2,j2)}$:

$$M = \begin{array}{|c|c|c|c|c|}\hline 1 & 9 & 6 & 3 & 7 \\ \hline 7 & 5 & 3 & 0 & 5 \\ \hline 0 & 7 & 6 & 5 & 3 \\ \hline 7 & 8 & 9 & 5 & 0 \\ \hline 9 & 5 & 3 & 7 & 8 \\ \hline \end{array}$$

En el intervalo $Q_{(2,2),(3,4)}$ el resultado es $5+3+0+7+6+5 = 26$.En el caso de 1D se definió $suma(x) = \sum_{k=1}^{x} V_k$, ahora esta debe tener dos dimensiones, es decir,$suma(i,j)$ debe contener la suma de los elementos en la submatriz $Q_{(1,1),(i,j)}$, entonces ahora se definirá:$suma(i,j) = \sum_{k=1}^{i}\sum_{w=1}^{j} M_{k,w}$, mas sin embargo pre-calcular $suma(i,j)$ de forma eficienterequiere de usar el principio de inclusión-exclusión, de manera trivial se puede calcular la primera fila como$suma(1,j) = \sum_{w=1}^{j} M_{1,w}$ y la primera columna como $suma(i,1) = \sum_{k=1}^{i} M_{k,1}$, en basea esto se puede calcular el resto de la matriz pero se debe tener algo de cuidado, por ejemplo si se toma$suma(2,2) = suma(1,2) + suma(2,1) + M_{2,2}$ se obtendría un resultado incorrecto pues se estaría realizando lasiguiente operación: $(M_{1,1} + M_{1,2}) + (M_{1,1}, M_{2,1}) + M_{2,2}$, se puede observar que el elemento$M_{1,1}$ se esta sumando dos veces, acá se aplica el principio de inclusión-exclusión:$|A| \cup |B| = |A| + |B| - |A \cap B|$ lo que significa que hace falta quitar la intersección, esta es$suma(1,1)$ entonce se puede generalizar como: $suma(i,j) = M_{i,j} + suma(i-1,j) + suma(i,j-1) - suma(i-1,j-1)$cuando $i,j \neq 1$.Una vez construido el pre-calculo es necesario realizar las consultas, se utilizara como ejemplo la consulta enel rango $Q_{(2,2),(3,4)}$, de igual manera se debe tener cuidado de no sumar un mismo intervalo mas de una vez,entonces para encontrar la suma en este intervalo se tomaría $suma(i2,j2)$ esta contendría$\sum_{k=1}^{i2}\sum_{w=1}^{j2} M_{k,w}$, esta tiene elemento adicionalescomo lo muestra la figura 1, al restarle

$suma(i1-1, j2)$ se quitan algunos elementos (figura 2), al restarle$suma(i2, j1-1)$ pasaríamos a restar dos veces el intervalo $M_{(1,1),(i1-1,j1-1)}$ (figura 3), por lo tanto esnecesario reponer lo faltante agregando $suma(i1-1, j1-1)$ (figura 4), de esta manera se puede generalizar laformula: $\sum_{k=i1}^{i2} \sum_{w=j1}^{j2} M_{k,w} = suma(i2, j2) - suma(i1-1, j2) - suma(i2, j1-1) + suma(i1-1, j1-1)$.Para

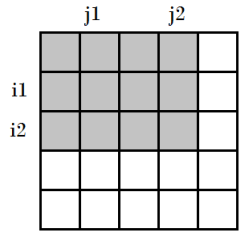


Figura 1.1

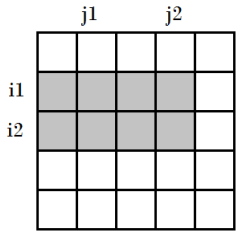


Figura 1.2

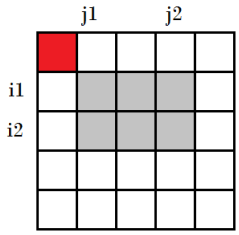


Figura 1.3

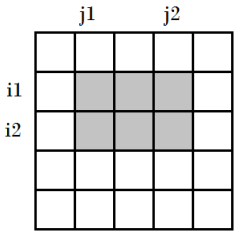


Figura 1.4

la solución en C++ la fila y columna cero de la tabla donde se guardara el pre-calculo se deberá llenarcon el elemento neutro, el cero, luego se debe llenar el resto de la tabla siguiendo las formulas antesestablecidas, este ejemplo es para consultas indexando desde 1.

```
1  void precalcular(){
2      memset(memo, 0, sizeof(memo));
3      for (int i = 1; i <= fila; i++)
4      for (int j = 1; j <= col; j++)
5          memo[i][j] = memo[i][j - 1] + memo[i - 1][j] +
6                          M[i - 1][j - 1] - memo[i - 1][j - 1];
7  }
8
9  int consulta(int f1, int c1, int f2, int c2){
10     return memo[f2][c2] - memo[f1-1][c2] -  memo[f2][c1-1] + memo[f1-1][c1-1];
11 }
```

Tablas aditivas en 3D

Las tablas aditivas se pueden generalizar para trabajar en n-dimensiones, mas sin embargo la dificultad de hacerlos pre-cálculos y las consultas aumenta bastante, pues crece considerablemente la cantidad de operaciones arealizar. En el caso 3D igualmente se debe tener cuidado con el principio de inclusión-exclusión, elpre-calculo se realizaría de la siguiente manera: sea$suma(i, j, k) = \sum_{x=1}^{i} \sum_{y=1}^{j} \sum_{z=1}^{k} V_{x,y,z}$, entonces$suma(i, j, k) = V_{i,j,k} + suma(i, j, k-1) + suma(i-1, j, k) + suma(i, j-1, k) - suma(i-1, j-1, k) - suma(i-1, j, k-1) - suma(i, j-1, k-1) + suma(i-1, j-1, k-1)$, y para las consultas: $\sum_{x=i1}^{i2} \sum_{y=j1}^{j2} \sum_{z=k1}^{k2} V_{x,y,z} = suma(i2, j2, k2) - suma(i2, j2, k1-1) - suma(i1-1, j2, k2) + suma(i1-1, j2, k1-1) - suma(i2, j1-1, k2) + suma(i2, j1-1, k1-1) + suma(i1-1, j1-1, k2) - suma(i1-1, j1-1, k1-1)$.Esta estructura de datos puede ser aplicada para otras operaciones que cumplan con las tres propiedades descritasanteriormente, la complejidad computacional de hacer el pre-calculo es lineal a la cantidad deelementos en la estructura, y las consultas se realizan en complejidad constante al realizar solamente operacionesdirectas sobre elementos en la tabla del pre-calculo.

1.2. Bibliografia

http://trainingcamp.org.ar/anteriores/2017/clases.shtml.
libro: competitive programming 3.

Capítulo 2

Grafos

2.1. Teoría de grafos

Los grafos son una estructura de datos donde se pueden relacionar distintos objetos entre si, los grafos se puedendefinir como un conjunto de nodos(objetos) unidos por aristas(relaciones), estos son estudiados por lamatemática y las ciencias de la computación, esta rama se conoce como teoría de grafos, ademas tienen muchasaplicaciones, por ejemplo permiten modelar redes informáticas, sistemas de carreteras, redes sociales, etc.

Clasificaciones generales

Los grafos se pueden clasificar de distintas formas, las mas utilizadas son:

1. **Grafos ponderados y no ponderados** Si las aristas de un grafo tienen un peso, es decir si paraatravesar una arista esta tiene un costo asociado, entonces el grafo se clasifica como ponderado, pero sininguna arista tiene algún costo entonces el grafo se clasifica como no ponderado.

2. **Grafos dirigidos y no dirigidos:** Si un grafo posee por lo menos una arista dirigida entoncesse clasifica como un grafo dirigido, una arista (A, B) es dirigida si esta permite el paso de A hacia$B$, pero no permite ir de B hacia A. Si todas las aristas son bidireccionales(no dirigidas) entoncesel grafo se clasifica como no dirigido.

3. **Grafos cíclicos y acíclicos** Se considera un grafo como acíclico cuando este no tiene ciclos, esdecir para cada pareja de nodos (A, B) si existe un camino para ir de A hacia B entonces no existeotro camino para ir de B hacia A. Si un grafo no es acíclico entonces este es cíclico.

4. **Grafos conexos y no conexos** Si para cada pareja de nodos (A, B) existe un camino para ir de$A$ hacia B y también existe camino para ir de B hacia A entonces el grafo es conexo, en casocontrario es un grafo no conexo.

Representación.

Los grafos se pueden representar de múltiples maneras cada una permite realizar distintos algoritmos, cada formade representar los grafos se puede ajustar según su tipo, un aspecto importante a considerar es que cada una aristabidireccional (A, B) se puede interpretar como dos aristas dirigidas (A, B) y (B, A). Existen principalmente detres formas distintas de representarlos:

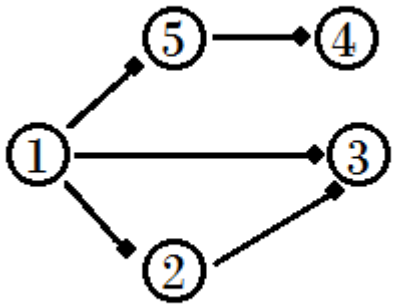
1. **Matriz de adyacencia:** Es una matriz M en la cual cada fila representa un nodo de inicio y cadacolumna un nodo destino(a cada nodo se le asignara un numero representando su posición en fila y columna),si existe una arista (A, B) entonces se marca la casilla $M_{A,B}$, en el caso de grafos ponderados sedebe llenar $M_{A,B}$ con el costo de la arista (A, B), para los todos los pares de nodos que no tenganaristas entre ellos el costo es infinito. Si el grafo es no ponderado entonces en la matriz simplemente semarca si existe o no la arista (A, B). Por definición cada nodo tiene conexión con sí mismo con costo cero.

2. **Lista de adyacencia:** Consiste en guardar para cada nodo una lista con las conexiones que posee,es decir, para cada arista (A, B) se pondrá en la lista de conexiones de A el nodo B. Si el grafo esponderado entonces se deberá guardar el nodo destino junto con su costo. Esta es la manera mas utilizadapara representar grafos, pues el espacio de memoria que ocupa es menor que el utilizado por una matriz deadyacencia y ademas facilita recorrer el grafo de manera sencilla.

3. **Lista de aristas:** Consiste en una lista en la cual se guardara todas las aristas de un grafo,para cada una se guarda el nodo de inicio, node de destino y el costo en caso de tener. Esta es la menosutilizada de las tres, pero esta facilita ordenar las aristas según su costo.

2.2. DFS y BFS

Los grafos son estructuras no lineales, estos no tienen un nodo inicio o un orden especifico para recorrerlos,existen principalmente dos algoritmos que permiten recorrer un grafo, estos no son algoritmos muy estrictos, osea se pueden modificar de múltiples maneras para realizar diferentes tareas, mas sin embargo la idea básica decada se debe mantener, para la implementacion de ambos se utiliza una lista de adyacencia, esta se representa comoun vector de vectores, por ejemplo:




```
vector<vector<int>> grafo(10);

void construir_grafo(){
    grafo[5].push_back(4);
    grafo[1].push_back(2);
    grafo[2].push_back(3);
    grafo[1].push_back(3);
    grafo[1].push_back(5);
}
```

Figura 2.1

DFS.

El DFS (deep first search) o Búsqueda en profundidad, es un algoritmo que permite recorrer un grafo, de maneragеneral consiste en tomar un nodo, marcarlo como visitado y para cada arista hacer un llamado recursivo al nododestino y repetir el proceso hasta que no queden mas nodos por visitar. Este es un algoritmo de backtraking conel cual se busca hacer una búsqueda completa por todos los nodos,

2.3. Bibliografia

https://es.wikipedia.org/wiki/Grafo
http://trainingcamp.org.ar/anteriores/2017/clases.shtml.
libro: competitive programming 3.