Estructuras de Datos

Matias Tealdi[1]

[1]Facultad de Matemática, Astronomía y Física
Universidad Nacional de Córdoba

Training Camp 2012

Contenidos

Conceptos

Estructura de Datos

es una forma particular de almacenar y organizar la información de mode que esta pueda ser utilizada eficientemente.

Ejemplos de estructuras conocidas:

- Arreglo
- Cola
- Pila
- . . .

Conceptos

Estructura de Datos

es una forma particular de almacenar y organizar la información de mode que esta pueda ser utilizada eficientemente.

Ejemplos de estructuras conocidas:

- Arreglo
- Cola
- Pila
- . . .

Principios Básicos

La implementación de las estructuras de datos requieren escribir un conjuntos de operaciones que manipulan la estructura. La eficiencia de la estructura debe ser analizada de acuerdo a estas operaciones.

Analicemos el caso particular de un arreglo y una lista enlazada.

- Inizializar la estructura, arreglo toma $O(n)$, lista enlazada toma $O(1)$
- Acceso a un elemento, arreglo $O(1)$, lista enlazada $O(n)$
- Insertar/eliminar un elemento, arreglo $O(n)$, lista enlazada $O(1)$ (si tengo la referencia del lugar a insertar/eliminar)

Principios Básicos

La implementación de las estructuras de datos requieren escribir un conjuntos de operaciones que manipulan la estructura. La eficiencia de la estructura debe ser analizada de acuerdo a estas operaciones.

Analicemos el caso particular de un arreglo y una lista enlazada.

- Inizializar la estructura, arreglo toma $O(n)$, lista enlazada toma $O(1)$
- Acceso a un elemento, arreglo $O(1)$, lista enlazada $O(n)$
- Insertar/eliminar un elemento, arreglo $O(n)$, lista enlazada $O(1)$ (si tengo la referencia del lugar a insertar/eliminar)

Contenidos

Union-Find

La estructura Union Find se utiliza para representar un conjunto de elementos acotado, particionado en un número de subconjuntos dijuntos.
Se pueden realizar las siguientes operaciones sobre una estructura Union-Find

- *Find*(x) : Determina a que subconjunto pertenece el elemento x.
- *Union*(ca, cb) : Une el subconjunto ca y cb. (subconjuntos dijuntos).

La implementación va a determinar la complejidad de estas funciones.

Contenidos

Implementación de Estructura Union-Find

```
1 #define N 1000
2 int comp[N];
3 void Init() { for(int i=0 ; i<N ; i++) comp[i] = i;}
4 int Find(int x) { return comp[x]; }
5 void Union(int x, int y) {
6     assert(Find(x) != Find(y));
7     int tmp = comp[x];
8     for(int i=0 ; i<N ; i++) if(comp[i] == tmp) comp[i] = comp[y];
9 }
```

Analisis de complejidad:

- Init : $O(n)$
- Union : $O(n)$
- Find : $O(1)$

Contenidos

Implementación de Estructura Union-Find

```
1  #define N 1000
2  int comp[N], rank[N];
3  void Init() { for(int i=0 ; i<N ; i++) comp[i] = i, rank[i] = 0; }
4  int Find(int x) { return comp[x] == x ? x : comp[x] = Find(comp[x]); }
5  void Union(int x, int y) {
6      assert(Find(x) != Find(y));
7      int cx = Find(x), cy = Find(y);
8      if(rank[cx] < rank[cy]) comp[cx] = cy;
9      else comp[cy] = cx;
10     if(rank[cx] == rank[cy]) rank[cx] ++;
11 }
```

Analisis de complejidad:

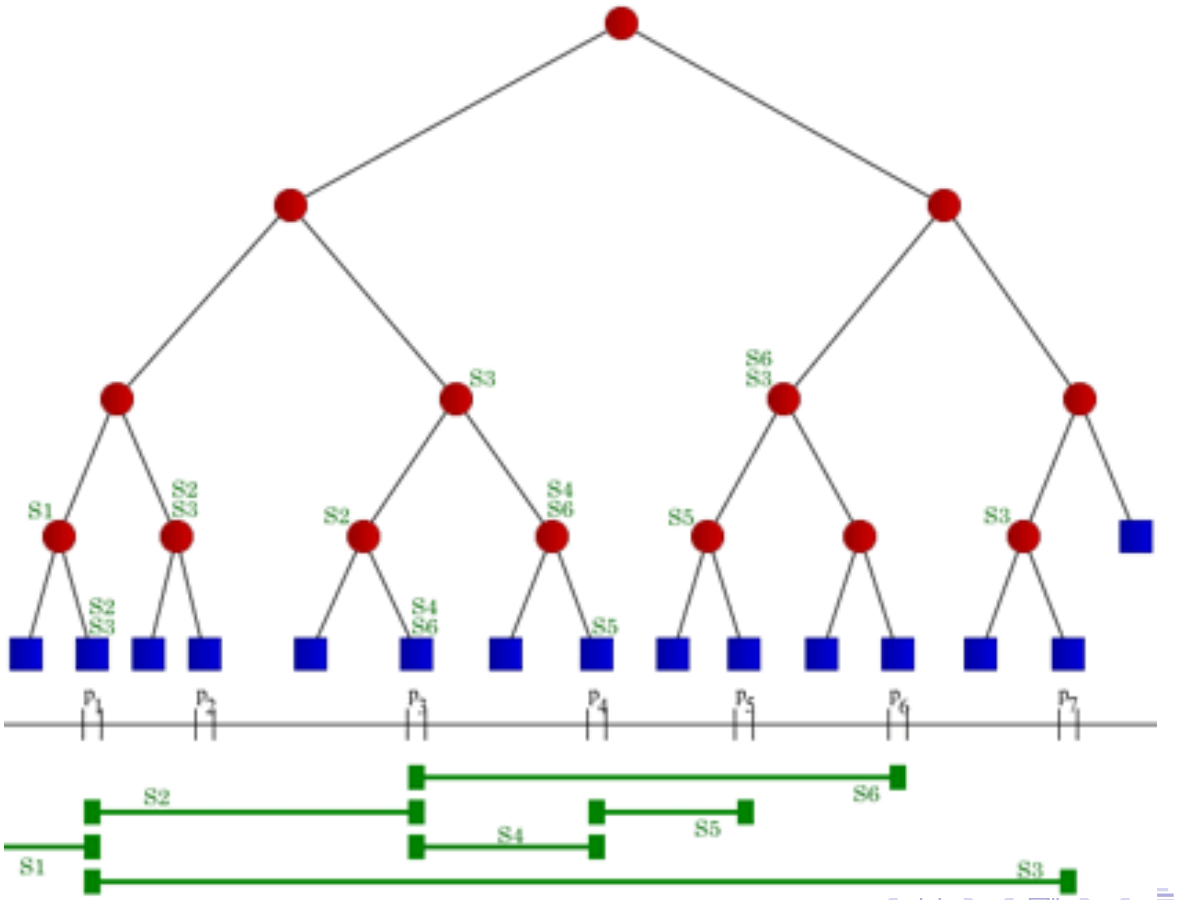
- Init : $O(n)$
- Union : $O(1)$
- Find : $\sim O(1)$

Contenidos

Principios Básicos

2 Un monton de cosas.

Un segment tree es una estructura de árbol binario en donde cada nodo representa un intervalo de un segmento. Esta estructura nos permite realizar consultas y actualizaciones en tiempos logarítmicos.



Contenidos

Range Minimum Query

Definición

Dado un arreglo $A[0..n)$ de n elementos, $RMQ(i, j)$ nos devuelve el mínimo/máximo elemento en el intervalo $[i, j]$ del arreglo A.

Una forma eficiente de resolver el problema de RMQ es utilizando la estructura de datos Segment tree. En la estructura, cada nodo mantiene mínimo/máximo del segmento que representa. Luego para consultar el mínimo/máximo de un segmento $[i, j]$, lo hacemos a travéz de la combinación de los valores que ya conocemos.

Range Minimum Query

Definición

Dado un arreglo $A[0..n)$ de n elementos, $RMQ(i, j)$ nos devuelve el mínimo/máximo elemento en el intervalo $[i, j]$ del arreglo A.

Una forma eficiente de resolver el problema de RMQ es utilizando la estructura de datos Segment tree. En la estructura, cada nodo mantiene mínimo/máximo del segmento que representa. Luego para consultar el mínimo/máximo de un segmento $[i, j]$, lo hacemos a travéz de la combinación de los valores que ya conocemos.

Contenidos

Implementación función Init

```
1  #define N
2  int tree[2*N];
3  int A[N];
4
5  void Init(int node, int a, int b) {
6      if(a == b) {
7          tree[node] = a;
8          return;
9      }
10     Init(node*2, a, (a+b)/2);
11     Init(node*2+1, (a+b)/2 + 1, b);
12
13     if(A[tree[node*2]] <= A[tree[node*2+1]]) tree[node] = tree[node*2];
14     else tree[node] = tree[node*2+1];
15 }
```

Implementación función get

```
1  int get(int node, int a, int b, int i, int j) {
2      // interseccion vacia
3      if(a>j || b<i) return -1;
4
5      //segmento [a,b] esta contenido en [i,j]
6      if(i<=a && b<=j) return tree[node];
7
8      int p1 = get(node*2, a, (a+b)/2, i, j);
9      int p2 = get(node*2+1, (a+b)/2+1, b, i, j);
10
11     if(p1 != -1 && p2 != -1) retrun A[tree[p1]] <= A[tree[p2]] ? p1 ? p2;
12     else if(p1 != -1) return p1;
13     return p2;
14 }
```

Implementación función update

```
1  void update(int node, int a, int b, int i) {
2      if(a==b && a == i) {
3          tree[node] = a;
4          return;
5      }
6      //i no esta en segmento [a,b]
7      if(i<a || i>b) return ;
8
9      update(node*2, a, (a+b)/2, i);
10     update(node*2+1, (a+b)/2+1, b, i);
11
12     if(A[tree[node*2]] <= A[tree[node*2+1]]) tree[node] = tree[node*2];
13     else tree[node] = tree[node*2+1];
14 }
```

Utilización

```
1  int main(void){
2      ...
3
4      // lleno el areglo A
5      Init(1, 0, n-1);
6
7      get(1, 0, n-1, i, j);
8
9      A[i] = x;
10     update(1, 0, n-1, i);
11
12     ...
13     return 0;
14 }
```

Contenidos

Analisis de Complejidad

Al ser un árbol binario, la profundidad del árbol es de a los sumo $log(n)$.

- Init : $O(2 * n)$
  Entro solo una ves a cada nodo, tengo 2*n nodos.
- get : $O(log(n))$
  Rocorro el árbol por solo una camino a alguna hoja.
- update : $O(log(n))$
  Rocorro el árbol por solo una camino a alguna hoja.

Contenidos

Sparse Table

Otra forma de solucionar el problema de RMQ es utilizando RMQ. La idea es preprocesar subarreglos de tamaño 2^k utilizando programación dinámica.

Mantenemos un arreglo de tamaño $M[0, N-1][0, LogN]$ donde $M[i][j]$ es el indice del valor mínimo del subarreglo que comienza en la posición i y tiene largo 2^j.

$$M[i][j] = \begin{cases} M[i][j-1] & \text{si } A[M[i][j-1]] \leq A[M[i+2^{j-1}-1][j-1]] \\ M[i+2^{j-1}-1][j-1] & \textit{sino}. \end{cases}$$

Consulta

Una ves que tenemos preprocesada la matriz M, veamos como podemos utilizarla para calcular $RMQ(i,j)$. La idea es utilizar dos bloques que cubran enteramente el intervalo $[i..j]$ y encontrar el mínimo entre ellos. Sea $k = [log(j-i+1)]$, para calcular el $RMQ(i,j)$ utilizamos la siguiente fórmula.

$$RMQ(i,j) = \begin{cases} M[i][k] & \text{si } A[M[i][k]] \leq A[M[j-2^k+1][k]] \\ M[j-2^k+1][k] & \textit{sino}. \end{cases}$$

Contenidos

Concepto

Árbol indexado binariamente

Es una estructura de árbol de segmentos que uliza la representación binaria de los números para guardar información. Nos permite realizar la consulta/actualización de valores acumulados sobre intervalos.

Un ejemplo muy común sería, dado un arreglo $A[1, n)$ de n elementos, la estructura nos permite realizar la consulta del valor acumulado en $A[i, j]$ y modificar un valor $A[i]$.

En este caso estamos haciendo la acumulada con la función suma, pero es importante saber que se puede utilizar cualquier función que cumpla con la propiedad asociativa y que tenga inverso.

Concepto

Árbol indexado binariamente

Es una estructura de árbol de segmentos que uliza la representación binaria de los números para guardar información. Nos permite realizar la consulta/actualización de valores acumulados sobre intervalos.

Un ejemplo muy común sería, dado un arreglo $A[1, n)$ de n elementos, la estructura nos permite realizar la consulta del valor acumulado en $A[i, j]$ y modificar un valor $A[i]$.

En este caso estamos haciendo la acumulada con la función suma, pero es importante saber que se puede utilizar cualquier función que cumpla con la propiedad asociativa y que tenga inverso.

Concepto

Árbol indexado binariamente

Es una estructura de árbol de segmentos que uliza la representación binaria de los números para guardar información. Nos permite realizar la consulta/actualización de valores acumulados sobre intervalos.

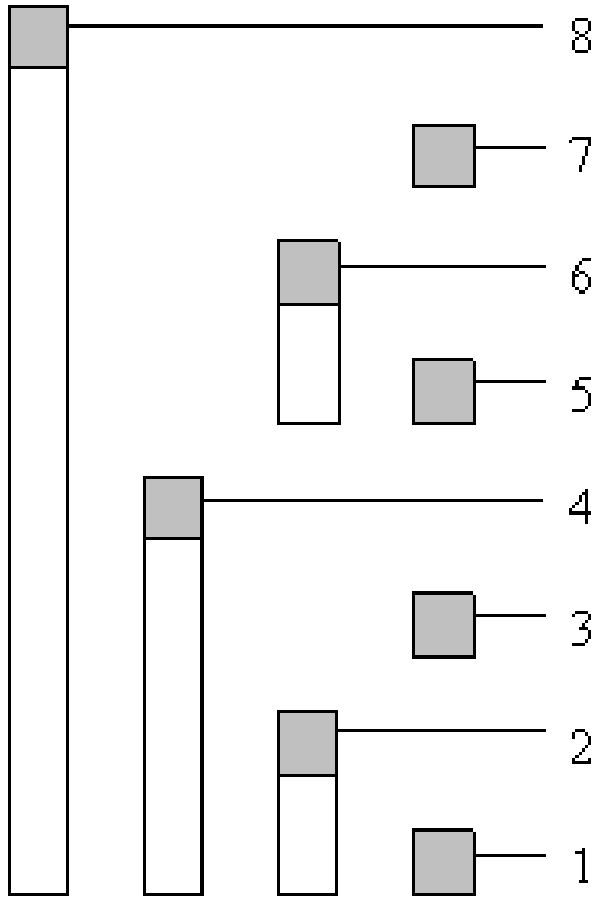
Un ejemplo muy común sería, dado un arreglo $A[1, n)$ de n elementos, la estructura nos permite realizar la consulta del valor acumulado en $A[i, j]$ y modificar un valor $A[i]$.

En este caso estamos haciendo la acumulada con la función suma, pero es importante saber que se puede utilizar cualquier función que cumpla con la propiedad asociativa y que tenga inverso.

Contenidos

Idea Básica

Los valores enteros pueden ser representados como sumatoria de potencias de dos. Del mismo modo, las frecuencias acumuladas pueden ser reprensetadas como sumas de subfrecuencias acumuladas.



Implementación

```
1  #define MAXN 1000000
2  int tree[MAXN];
3  void Init(){ for(int i=0 ; i<MAXN ; i++) tree[i] = 0; }
4  int get(int ind) {
5      int sum = 0;
6      while(ind) {
7          suma += tree[ind];
8          ind -= ind & (-ind);
9      }
10     return ind;
11 }
12 void update(int ind, int valor) {
13     while(ind <= MAXN) {
14         tree[ind] += valor;
15         ind += ind & (-ind);
16     }
17 }
```

Contenidos

Analisis de Complejidad

Como utilizamos la representación binaria de los enteros

- Init : $O(n)$
  Solo tengo que inicializar la memoria.
- get : $O(log(n))$
  Recorro la representación binaria de la acumulada que quiero consultar.
- update : $O(log(n))$
  Recorro la representación binaria de la acumulada que quiero actualizar.