Tópicos Avançados em Algoritmos

Hamilton José Brumatto

Bacharelado em Ciências da Computação - UESC

16 de maio de 2019

Árvore Binária Indexada

A árvore

- A árvore binária indexada (Binary Indexed Tree - BIT), também conhecida como árvore de Fenwick foi inventada por Peter M. Fenwick in 1994
- Esta árvore é útil para implementar uma estrutura de dados que acumula de forma dinâmica tabelas de frequências.
- Vamos considerar um exemplo onde temos $m = 11$ alunos que realizam um teste, onde cada aluno pode receber notas de 1 a 10 (em valores inteiros).
- Dada a distribuição de notas de cada alunos, queremos saber a frequência de cada nota.
- Conhecida a frequência de cada nota queremos saber a frequência em um intervalo de notas, por exemplo, quantos alunos tiveram notas entre 4 e 7.

Exemplo

notas = {2,4,5,5,6,6,6,7,7,8,9}

Nota	freq. (f)	Acumulado (cf)	Comentários
0	-	-	Ignorado (valor sentinela)
1	0	0	cf[1] = f[1] = 0
2	1	1	cf[2] = f[2]+f[1]
3	0	1	cf[3] = f[3]+f[2]+f[1]
4	1	2	cf[4] = f[4]+cf[3]
5	2	4	cf[5] = f[5]+cf[4]
6	3	7	cf[6] = f[6]+cf[5]
7	2	9	cf[7] = f[7]+cf[6]
8	1	10	cf[8] = f[8]+cf[7]
9	1	11	cf[9] = f[9]+cf[8]
10	0	11	cf[10] = f[10]+cf[9]

RSQ - Range Sum Query (Busca da Soma de um intervalo)

- A tabela de frequência acumulada pode ser usada como soluçao para o problema RSQ, que é a busca da soma de um intervalo.
- Para $n = 10$, `RSQ(1,1) = 0`, ..., `RSQ(1,6) = 7`, ..., `RSQ(1,8) = 10`, ..., e `RSQ(1,10) 11`.
- Também é possível obter o RSQ de um intervalo qualquer: `RSQ(i,j) = RSQ(1,j) - RSQ(1,i-1)`
- `RSQ(4,7) = RSQ(1,7) - RSQ(1,3)`
- Se as frequências forem estáticas, esta tabela é suficiente para resolver o problema, em $O(n)$ construímos a tabela, e qualquer consulta é $O(1)$.
- Se as frequências forem dinamicamente alteradas qualquer atualização é $O(n)$. Outra estrutura dinâmica é melhor para resolver o problema. Uma árvore de segmentos pode ser usada.
- Uma solução mais simples é a Árvore de Fenwick.

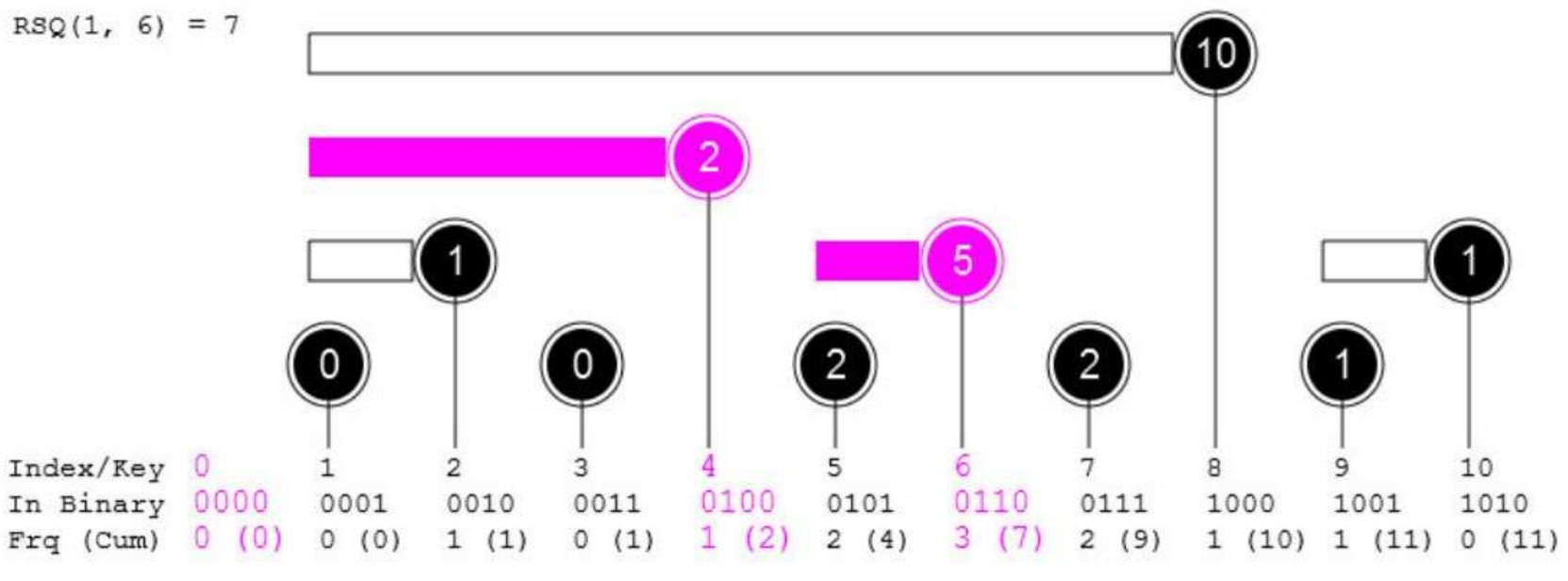
A Árvore de Fenwick

- Vamos considerar a operação LSOne(i) = $i\&(-i)$ como a operaçao que retorna o bit 1 menos significante de um número.
- A árvore de Fenwick é implementada na forma de um vetor.
- A árvore de Fenwick é uma árvore indexada pelos bits de suas chaves inteiras (por isso BIT-Binary Indexed Tree).
- As chaves de inteiros estão no intervalo $[1..n]$, exclui-se o índice 0.
- No exemplo anterior, os inteiros são as notas, no intervalo [1..10].

A Árvore de Fenwick

- Na árvore, o índice i é responsável pelos elementos no intervalo $[i - LSOne(i) + 1..i]$, ou para uma melhor visualização, no intervalor $(i - LSOne(i)..i)$.
- Por exemplo, o índice $6 = 0110$ é responsável pelos elementos do intervalo $(0100..0110]$, ou seja $(4..6]$, $[5..6]$
- Assim sendo a árvore representada pelo vetor ft, $ft(6) = 5 = f(5) + f(6)$
- Uma $RSQ(1..6)$ seria obtido com $ft(6) + ft(LSOne(6) + ft(LSOne(LSOne(6))$, até que a chamada recursiva de $LSOne()$ retorne 0, que não será computado: $RSQ(1..6) = ft(6) + ft(4) = ft(0110) + ft(0100)$.

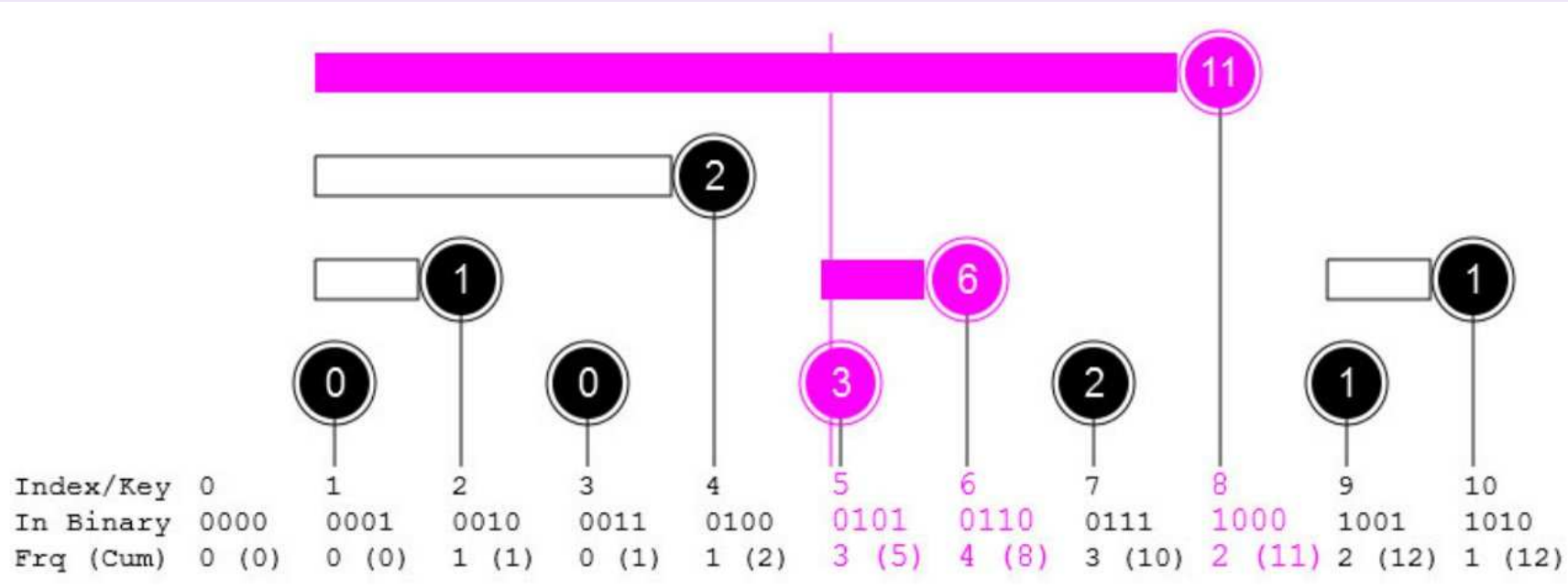
A Árvore de Fenwick



A Árvore de Fewnwick

- Ao atualizar um determinado índice, é necessário atualizar, também, os índices superiores.
- O índice superior de um índice i é obtido como: $i + LSOne(i)$.
- A atualização é feita para todos índices i até que $i > n$.
- Atualizar o índice $5 = 0101$ implica atualizar também o índice $6 = 0101 + 0001 = 0110$, o índice $8 = 0110 + 0010 = 1000$.

A Árvore de Fenwick



A Árvore de Fenwick

- A árvore binária de índice realiza busca RSQ e atualização de elementos em tempo $O(\log n)$ e em espaço em $O(n)$.
- É ideal para problemas com atualização dinâmica de valores, se os valores forem estáticos, construímos uma tabela em $O(n)$ com busca em $O(1)$, como vimos no início.
- Para m chaves no intervalo $[1..n]$ podemos comparar:

Ação	**Árv. Segmentos**	**Árv. Fenwick**
Construir a árvore	$O(n)$	$O(m \log n)$
Busca dinâmica RMin/MaxQ	OK	OK
Complexidade Busca	$O(\log n)$	$O(\log n)$
Complexidade Atualização	$O(\log n)$	$O(\log n)$
Comprimento do código	longo	curto

Código: Árvore de Fenwick class ft

```
class FenwickTree {
private:  vector<int> ft;
public:
 FenwickTree(int n) { ft.assign(n + 1, 0); } // inicia com n + 1 zeros
 int rsq(int b) { // retorna RSQ(1, b)
  int sum = 0;
  for (; b; b -= LSOne(b)) sum += ft[b];
  return sum;
 } // nota:  LSOne(S) (S & (-S))
 int rsq(int a, int b) { return rsq(b) - (a == 1 ?  0 :  rsq(a - 1)); }
 void adjust(int k, int v) {
  // atualiza o valor do k-ésimo elemento em v (positivo ou negativo)
  for (; k < (int)ft.size(); k += LSOne(k)) ft[k] += v;
 }
};
```

Problema 1804 - URI: Precisa-se de Matemáticos em Marte

Aconteceu na semana passada em Beberibe, CE, o VIII Latin-American Algorithms, Graphs and Optimisation Symposium (LAGOS 2015), uma conferência que reuniu matemáticos e cientistas da Computação do mundo todo para discutirem alguns desafios computacionais. A fim de que os participantes pudessem relaxar um pouco, os organizadores do evento contrataram buggies para levar os pesquisadores a alguns pontos turísticos da região. Os buggies saíram do hotel numa fila, um atrás do outro. Quem conhece os passeios de buggy do Ceará sabe que o turista pode pedir ao bugueiro um passeio com emoção ou sem emoção. No entanto, a excursão do LAGOS contou com uma emoção extra. Como em Marte atualmente há uma carência de pesquisadores em Algoritmos, Grafos e Otimização, alienígenas marcianos começaram a abduzir alguns buggies, com todas as pessoas a bordo (inclusive o pobre bugueiro, que nada tinha a ver com a história). Foi muito triste nosso planeta ter perdido mentes tão brilhantes para Marte, mas a tragédia teria sido menor se cada bugueiro mantivesse atualizada a informação de quantas pessoas no total estavam nos buggies atrás dele - assim ele poderia perceber a aproximação da sonda alienígena e acelerar o buggy.

Problema 1804: Entrada e Saída

Entrada:
Um inteiro **N** ($1 \leqslant \mathbf{N} \leqslant 10^5$) constitui a primeira linha da entrada, o qual representa o número de buggies que deixaram o hotel. A segunda linha da entrada é constituída por **N** inteiros $\mathbf{p}_i$ ($1 \leqslant \mathbf{p}_i \leqslant 5$, $1 \leqslant \mathbf{i} \leqslant \mathbf{N}$), cada um representando o número de pessoas no buggy **i** (incluindo o bugueiro). Cada uma das linhas seguintes pode ser constituída:

- por um caractere 'a' seguido de um inteiro **i** ($1 \leqslant \mathbf{i} \leqslant \mathbf{N}$), o qual caracteriza a abdução do buggy **i** (que ainda não havia sido abduzido);
- por um caractere '?' seguido de um inteiro **i** ($1 \leqslant \mathbf{i} \leqslant \mathbf{N}$), o qual caracteriza que, naquele momento, o bugueiro do (ainda) não abduzido buggy **i** gostaria de saber quantas pessoas da excursão restavam atrás do seu buggy.

Considere que o identificador **i** de um buggy não muda. Assim, o terceiro buggy a deixar o hotel deverá sempre ser identificado pelo inteiro 3, mesmo que os buggies 1 e 2 sejam abduzidos.

Saída:
Para cada linha "? `i`", imprima uma linha contendo o número de pessoas da excursão ainda remascentes atrás do buggy **i** naquele momento. Mas não seja bugueiro (nada de pôr bugs em seu código)!