Hash

Hash universal e *hash* perfeito

Prof. Edson Alves - UnB/FCTE

Sumário

Hash universal

Hash universal

- Qualquer que seja a função de *hash* h, é possível construir uma sequência de chaves $K_1, K_2, \ldots, K_N$ tais que $h(K_i) = h(K_j)$
- Esta sequência levaria ao pior caso da inserção e da busca, com complexidade $O(T)$
- A ideia do *hash* universal é a mesma do *quicksort*: escolher, no início da execução do algoritmo, uma função de *hash* dentre uma família de *hashes* possíveis
- Deste modo, diferentes execuções do algoritmo levariam a resultados diferentes, mesmo para uma entrada fixa
- Assim, uma única sequência não seria capaz de levar ao pior caso em todas as execuções, melhorando a performance no caso médio

Conjunto universal

- Seja $\mathcal{H}$ um conjunto de funções de *hash* que mapeiam as chaves no intervalo $[0, T-1]$
- O conjunto $\mathcal{H}$ é dito universal de *hashes* se, para todos os pares de chaves distintas K e L, o subconjunto $S_{KL} \subset \mathcal{H}$ tal que

$$S_{KL} = \{f, g \in \mathcal{H} \mid f \neq g \ \text{ e } \ f(K) = g(L)\}$$

  tem tamanho $|S_{KL}| \leq |\mathcal{H}|/T$
- Deste modo, escolhida aleatoriamente uma função $h \in \mathcal{H}$, a probabilidade existam uma colisão entre duas chaves $K_1 \neq K_2$ é igual a

$$P(h(K_1) = h(K_2)) = \frac{|S_{K_1 K_2}|}{|\mathcal{H}|} \leq \frac{|\mathcal{H}|/T}{|\mathcal{H}|} = \frac{1}{T}$$

Exemplo de conjunto universal

- Seja p um número primo tal que o valor de qualquer chave K seja menor do que p
- Seja $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$ e $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$
- Defina

$$h_{ab}(K) = (aK + b \pmod{p}) \pmod{T},$$

- É possível demonstrar que

$$\mathcal{H}_{pT} = \{h_{ab} \mid a \in \mathbb{Z}_p^* \ \text{ e } \ b \in \mathbb{Z}_p\}$$

  é um conjunto universal de *hashes* com $|\mathcal{H}_{pT}| = p(p-1)$ elementos
- Observe que não há restrições impostas ao tamanho da tabela T

Exemplo de uso do conjunto universal

Chaves a serem inseridas: $33, 17, 95, 27, 88, 15, 54, 62, 40$

Tamanho da tabela: $T = 20$, $p = 101$

K	$h_{1,0}(K)$	$h_{2,1}(K)$	$h_{44,37}(K)$	$h_{51,97}(K)$	$h_{5,11}(K)$
33	13	7	15	**3**	15
17	17	**15**	**18**	15	16
95	**15**	10	16	**14**	2
27	7	**15**	13	**0**	5
88	8	16	**11**	**0**	7
15	**15**	11	**11**	**14**	6
54	14	8	10	**3**	19
62	2	4	**18**	7	18
40	0	1	0	16	9

Implementação de conjunto universal de hashes

```
#include <bits/stdc++.h>

using namespace std;

template<typename I, size_t T>
class HashSet {
private:
    size_t mod(const I& a, int b) {
        auto r = a % b;
        return r < 0 ? r + b : r;
    }

    size_t h(const I& K) { return mod(a*K + b, p); }
    size_t N(const I& K, size_t i) { return mod(h(K) + i, T); }

    vector<I> xs;
    I p, a, b;
    bitset<T> used;
```

Implementação de conjunto universal de hashes

```
public:
    HashSet(const I& pv) : xs(T), p(pv), a(rand() % (p - 1) + 1), b(rand() % p) {}

    bool insert(const I& K)
    {
       if (used.count() == T)
            return false;

        for (size_t i = 0; i < T; ++i) {
            auto pos = N(K, i);

            if (not used[pos]) {
                xs[pos] = K;
                used[pos] = true;
                break;
            }
        }

        return true;
    }
```

Hash perfeito

Definição de *hash* perfeito

- Seja um conjunto de chaves $\mathcal{K} = \{K_1, K_2, \ldots, K_N\}$, a serem inseridas em uma tabela com tamanho T
- Uma função $h : \mathcal{K} \to [0, T-1]$ é um *hash* perfeito para $\mathcal{K}$ se para todos os pares de índices (i, j), com $i \neq j$, segue que $h(K_i) \neq h(K_j)$
- Veja que a definição de *hash* perfeito depende do conjunto $\mathcal{K}$
- Existem T^N funções $h : \mathcal{K} \to [0, T-1]$
- Destas, apenas

$$A_{T,N} = \frac{T!}{(T-N)!}$$

  são *hashes* perfeitos
- Por exemplo, para $T = 100, N = 80$, há $100^{80} = 10^{160}$ funções, dentre as quais $A_{100,80} < 10^{140}$ são *hashes* perfeitos
- Logo, uma a cada 10^{20} destas funções serão *hashes* perfeitos

Construção de um *hash* perfeito

- Embora exista, em valor absoluto, um grande número de *hashes* perfeitos, não é tarefa trivial determinar um deles na prática
- A maior não tem sequer uma representação óbvia como função
- Pode-se construir um *hash* perfeito para o conjunto $\mathcal{K}$ combinando-se três ideias já apresentadas: encadeamento, *hash* duplo e *hash* universal
- O *hash* universal é utilizado para determinar o tamanho das listas encadeadas associadas a cada entrada da tabela, segundo o teorema abaixo

Teorema

Seja $T = N^2$ e $h \in \mathcal{H}_{pT}$. Então a probabilidade de que exista colisão entre duas chaves distintas de $\mathcal{K}$ é inferior a $1/2$

Construção de um *hash* perfeito

- Se o valor $T = N^2$ for pequeno, é possível encontrar um *hash* perfeito em $\mathcal{H}_{pT}$ após algumas tentativas
- Porém, para valores grandes de T, a ideia é utilizar o encadeamento com *hash* duplo
- Escolha uma função de *hash* h no conjunto universal $\mathcal{H}_{pN}$
- Seja n_j o número de chaves K em $\mathcal{K}$ tais que $h(K) = j$, com $j = 0, 1, 2, \ldots, N - 1$
- A ideia é associar uma nova tabela t_j, de tamanho n_j^2, para cada célula de uma tabela de tamanho N
- Os elementos que colidiram na célula j são então mapeados em T_j, através de uma nova função de *hash* $\hat{h} \in \mathcal{H}_{qn_j^2}$, onde q é um primo maior do que n_j^2

Construção de um *hash* perfeito

- Embora a abordagem descrita leve a crer que o espaço ocupado por todas as tabelas auxiliares t_i seja $O(N^2)$, o teorema abaixo mostra que, de fato, o espaço em memória é proporcional a N
- Desta forma, é possível construir um *hash* perfeito onde a função h localiza a célula j da tabela principal onde a chave se encontra, e sua posição exata na tabela auxiliar t_j é dada pela função $\hat{k}$, com memória $O(N)$

Teorema

Seja $h \in \mathcal{H}_{pN}$ uma função de *hash* e n_j o número de chaves K em $\mathcal{K}$ tais que $h(K) = j$, com $j = 0, 1, \ldots, N-1$. Então

$$E\left[\sum_{j=0}^{N-1} n_j^2\right] < 2N$$

Exemplo de *hash* perfeito

Parâmetros: $N = 10, p = 101, q = 103$

Hash: $h(K) = h_{1,0}(K) = K \pmod{10}$

j	a	b	n_j^2	
0	-	-	0	
1	0	0	1	71
2	-	-	0	
3	-	-	0	
4	-	-	0	
5	2	1	4	65, , , 25
6	-	-	0	
7	99	3	9	97, , 17, , , , , 37,
8	44	37	16	, 28, , , , , 78, , , 48, , , , , 58,
9	-	-	0	

Referências

1. **CORMEN**, Thomas H.; **LEISERSON**, Charles E.; **RIVEST**, Ronald L.; **STEIN**, Clifford. *Introduction to Algorithms*, The MIT Press, 3rd edition, 2009.
2. **DROZDEK**, Adam. *Algoritmos e Estruturas de Dados em C++*, 2002.
3. **RADKE**, Charles E. *The Use of Quadratic Residue Research*, Communications of the ACM, volume 13, issue 2, pg 103–105, 1970[1].
4. **STROUSTROUP**, Bjarne. *The C++ Programming Language*, 2013.
5. C++ Reference[2].

[1] https://dl.acm.org/citation.cfm?id=362036
[2] https://en.cppreference.com/w/