Chương 8: Kỹ thuật thỏa hiệp không-thời gian

Giới thiệu chung

Thỏa hiệp không-thời gian là tiếp cận sử dụng thêm một lượng tài nguyên nhất định của máy tính để lưu trữ hoặc tổ chức lại dữ liệu, nhằm tăng cường tốc độ thực thi khi xử lý vấn đề liên quan.

Ví dụ:

- Lính canh
- Bất biến vòng lặp

Thỏa hiệp không-thời gian có thể được chia thành hai nhóm chính sau:

Nâng cấp dữ liệu đầu vào (*input enhancement*)

Dữ liệu đầu vào sẽ được tiền xử lý một phần hay toàn bộ. Quá trình tiền xử lý sẽ cố gắng tìm kiếm, phát hiện thông tin ẩn bên trong dữ liệu.

Tiền cấu trúc dữ liệu đầu vào (*prestructuring*)

Tái cấu trúc hay phân bố lại tổ chức của dữ liệu ban đầu cho phù hợp với xử lý sau này.

Chú ý:

- Khi sử dụng kỹ thuật "thỏa hiệp không-thời gian", hai loại tài nguyên (thời gian và không gian) không cạnh tranh lẫn nhau.
- Qui hoạch động là một nhánh của chiến lược thiết kế này.

Nâng cấp dữ liệu đầu vào

Sắp xếp đếm (Countingsort)

Ý tưởng: Tìm xem một phần tử trong mảng cần sắp sẽ đứng ở vị trí nào trong mảng được sắp bằng cách đếm xem có bao nhiêu phần tử đứng trước nó.

Giải thuật

```
CountingSort(a[1 .. n]) {
   count[1 .. n] = 0;
   for (i = 1; i ≤ n – 1; i++)
      for (j = i + 1; j ≤ n; j++)
         if (a[i] < a[j])
            count[j]++;
         else
            count[i]++;
   for (i = 1; i ≤ n; i++)
      b[count[i]] = a[i];

   a[1 .. n]   = b[1 .. n];
}
```

Distribution counting

Cho rằng miền giá trị của các phần tử thuộc về đoạn [0 .. *high*].

Giải thuật

```
DistributionCounting(a[1 .. n]) {
   f[0 .. high] = 0;
   for (i = 1; i ≤ n; i++)
      f[a[i]] ++;
   for (i = 1; i ≤ high; i++)
      f[i] += f[i – 1];

   for (i = n; i ≥ 1; i--) {
      b[f[a[i]]] = a[i];
      f[a[i]]--;
   }
   a[1 .. n] = b[1 .. n];
}
```

Mở rộng

Có thể tiết kiệm vùng nhớ hơn nếu ta biết được miền giá trị của các phần tử thuộc về đoạn [*lo* .. *hi*] nào đó. Từ đây, mảng chứa tần số sẽ là: `f[0 .. hi-lo]`.

Tìm kiếm chuỗi – Giải thuật Horspool

Phát biểu: Tìm kiếm sự xuất hiện của chuỗi P chiều dài *m* trên chuỗi T chiều dài *n*.

Ý tưởng: Đầu tiên, *P* sẽ được gióng thẳng hàng với *T* và ký hiệu cuối cùng của *P* sẽ được so sánh với ký hiệu tương ứng trên *T*. Nếu trùng nhau thì đi tiếp sang *trái* cho đến khi toàn bộ *P* được xử lý thì thông báo tìm thấy, ngược lại thì chuyển dời *P* sang phải với bước chuyển *s* nào đó.

Giải thuật sẽ xác định *s* bằng cách dựa vào ký tự *c* trên *T* đang gióng thẳng hàng với ký tự cuối cùng của *P*.

Quá trình tiền xử lý sẽ tính toán *khoảng cách chuyển dời* (dựa trên *pattern*) và lưu thông tin vào mảng *D*. Kích thước mảng *D* là |Σ| và tương ứng với mỗi ký tự *c*, *D*[*c*] được tính theo công thức sau:

$$\forall c \in \Sigma, D[c] = \begin{cases} m & c \notin P[1..m-1] \\ khoảng\ cách\ từ\ c\ phải\ nhất\ đến\ P[m] & c \in P[1..m-1] \end{cases}$$

Ví dụ: *P* = FEDERER với *m* = 7 và *text* chỉ chứa các ký tự tiếng Anh cùng khoảng trắng (biểu diễn bởi '▫').

Ký tự *c*	A	B	C	D	E	F	…	R	…	Y	Z	▫
Khoảng chuyển	7	7	7	**4**	**1**	**6**	7	**2**	7	7	7	7

Giải thuật

```
ComputeArray(P[1..m], D, Σ) {
   for (Mỗi ký tự c ∈ Σ)
      D[c] = m;
   for (i = 1; i ≤ m - 1; i++)
      D[P[i]] = m – i;
}
Horspool(P[1..m], T[1..n], Σ) {
   int D[|Σ|];
   ComputeArray(P, D, Σ);
   i = m;
   while (i ≤ n) {
      k = 0;
      while (k < m) && (P[m - k] == T[i - k])
         k++;
      if (k == m)
         print "Mẫu xuất hiện tại i – m + 1";
      i += D[T[i]];
   }
}
```

Tiền cấu trúc dữ liệu đầu vào

Bảng băm

Băm (*hashing*) là hành vi phân bố các khóa vào mảng một chiều $H[0 .. m - 1]$, gọi là *bảng băm* (*hash table*). Sự phân bố được thực hiện thông qua một *hàm băm h* (*hash function*), có nhiệm vụ gán một số nguyên trong đoạn 0 và $m - 1$ (gọi là địa chỉ băm – *hash address*) cho mỗi khóa.

Ví dụ: Nếu khóa là số nguyên không âm, hàm băm có thể là $h(K) = K \bmod m$.

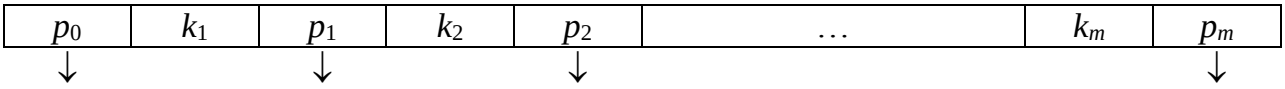
Khi làm việc với kỹ thuật này, có hai vấn đề cần quan tâm:

- Hàm băm cần phải phân bố các khóa trong bảng băm một cách đồng đều nhất có thể.
- Khả năng xử lý đụng độ (*collision*).

Có hai cách quản lý bảng băm, đó là Băm mở (*open hashing* hay *separate chaining*) và Băm đóng (*open addressing*) với những cách xử lý đụng độ riêng. Trong đó, Băm mở được xem là thông dụng hơn.

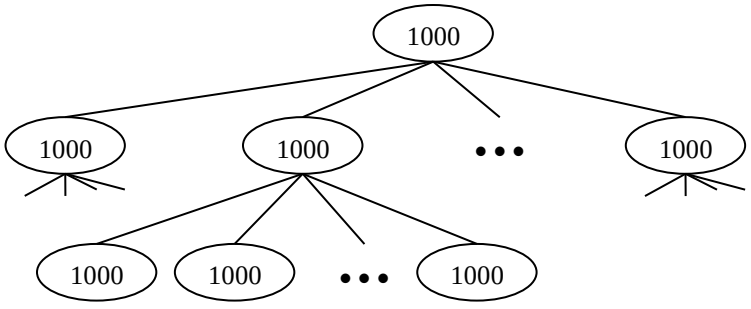
B-cây

Một nút của B-cây có cấu tạo như sau:



với k_i ($i \in [1, m]$) là các khóa ($k_i < k_{i+1}$) và p_j ($j \in [0, m]$) là các con trỏ đến những nút con.

Một nút có k khóa thì nó sẽ quản lý $(k + 1)$ nút con. Xét B-cây với $k = 1000$.



1 nút
(1000 khóa)

1 × 1001 nút
(1000 × 1001 =
1,001,000 khóa)

1001 × 1001 =1002001 nút
(1000 × 1002001 =
1,002,001,000 khóa)

Với chiều cao là 2, cây có thể lưu hơn 1 tỉ khóa.

Định nghĩa: B-cây có bậc $t \geq 2$, chiều cao $h > 0$ gồm:

- Một nút gốc, chứa từ 1 đến $2t$ khóa.
- Nút trong hoặc nút lá: chứa từ t đến $2t$ khóa.
- Mọi nút lá đều có cùng mức (*level*).
- Nếu gọi m là tổng số khóa của một nút (không phải nút lá) thì nó có $m + 1$ nút con.