TRƯỜNG ĐẠI HỌC ĐÀ LẠT
KHOA TOÁN - TIN HỌC

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT 2

TRƯƠNG CHÍ TÍN

Đà Lạt 2010

MỤC LỤC

LỜI NÓI ĐẦU

Giáo trình này nhằm cung cấp cho sinh viên các kiến thức nâng cao về cấu trúc dữ liệu và các thuật toán có liên quan. Để có thể nắm bắt các kiến thức trình bày trong giáo trình, sinh viên cần nắm được các kiến thức về tin học đại cương, kỹ thuật lập trình, nhập môn cấu trúc dữ liệu và thuật toán. Các kiến thức này sẽ tạo điều kiện cho sinh viên học tiếp các kiến thức về kỹ thuật lập trình nâng cao, đồ họa, trí tuệ nhân tạo, ...

Nội dung giáo trình gồm 3 chương:

- Chương 1: Giới thiệu các cách tổ chức file theo kiểu tuần tự và chỉ mục, cùng với các thuật toán sắp xếp trộn trên file.
- Chương 2: Trình bày một loại cây nhiều nhánh đặc biệt là B – cây. Nó có nhiều ứng dụng trong việc lưu trữ và tìm kiếm trên các bộ dữ liệu lớn.
- Chương 3: Giới thiệu các phương pháp tìm kiếm hiệu quả trên các bộ dữ liệu lớn dựa vào bảng băm: phương pháp băm liên kết và băm theo địa chỉ mở.

Chắn chắn rằng trong giáo trình sẽ còn nhiều khiếm khuyết, tác giả mong muốn nhận được và rất biết ơn các ý kiến quí báu đóng góp của đồng nghiệp cũng như bạn đọc để giáo trình này có thể hoàn thiện hơn nữa về mặt nội dung cũng như hình thức trong lần tái bản sau.

Đà lạt, 10/2009

Tác giả

Chương 1

TẬP TIN

I/. Giới thiệu tập tin

I.1. Định nghĩa tập tin (file)

Tập tin là tập các thông tin về các đối tượng nào đó *có quan hệ với nhau*, được *lưu trữ* thành *một đơn vị* trên *bộ nhớ ngoài*.

Trên thực tế, ta thường dùng tập tin để *lưu lâu dài* thông tin với *số lượng lớn*. Các phương pháp sắp xếp và tìm kiếm được giới thiệu trong giáo trình "Cấu trúc dữ liệu và thuật toán 1" được áp dụng khi lượng dữ liệu không lớn lắm được lưu giữ ở bộ nhớ trong. (Các thao tác sơ cấp trên các kiểu tập tin chính trong C++ được giới thiệu trong phần phụ lục).

I.2. Tổ chức tập tin

Dựa trên các thao tác sơ cấp truy nhập file trên đây, ta có thể xây dựng các thuật toán phức tạp và hữu ích khác trên ***file chứa các đối tượng có cùng cấu trúc***. Khi xét đến độ hiệu quả của các thuật toán này (đặc biệt về mặt thời gian), ta có thể tổ chức file theo 2 kiểu: tuần tự hay có chỉ mục.

* Khi lưu và truy cập các đối tượng theo kiểu tuần tự trong một file, ta có kiểu ***file tuần tự***.

Ví dụ 1: Giả sử ta cần lưu các đối tượng A, C, B cùng kiểu T vào file *f*.

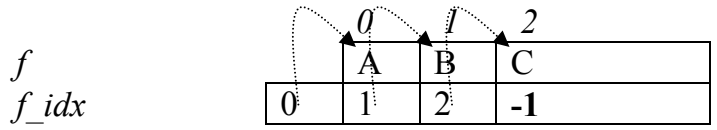
Tuy cách lưu trữ này rất đơn giản khi cài đặt nhưng ta sẽ gặp *nhiều nhược điểm (về mặt thời gian)* khi gặp các tình huống sau. Nếu ta cần chèn thêm 1 đối tượng D vào trước A thì ta phải dời mọi phần tử từ A qua phải một vị trí; nếu ta muốn xóa đối tượng A, thì ta phải dời mọi phần tử từ ngay sau A qua trái một vị trí. Đối với các *tập tin lưu nhiều đối tượng* có cùng kiểu dữ liệu T (trên thực tế, ta thường gặp trường hợp *T có kích thước (bytes) lưu trữ lớn*), nếu phải *dùng nhiều thao tác chèn và xóa* sẽ *mất rất nhiều thời gian* cho việc *dời chỗ* các phần tử.

Tương tự như các cấu trúc dữ liệu được cài đặt ở bộ nhớ trong, các hạn chế này cũng xuất hiện trong kiểu dữ liệu *mảng;* để khắc phục chúng, ta có thể dùng kiểu danh sách liên kết. Để khắc phục nhược điểm trong các thao tác chèn, xóa trên kiểu file tuần tự, ta có thể tổ chức tập tin theo *kiểu chỉ mục đơn giản (tương tự như danh sách liên kết)* như sau:

* Khi cần lưu một dãy các đối tượng có cùng kiểu T vào file *f*, ngoài việc dùng file *f* như cách tổ chức tuần tự như trên, ta *dùng kèm thêm* một ***file chỉ mục f_idx*** tương ứng để *chứa các địa chỉ (hay thứ tự) của các đối tượng thực sự trong*

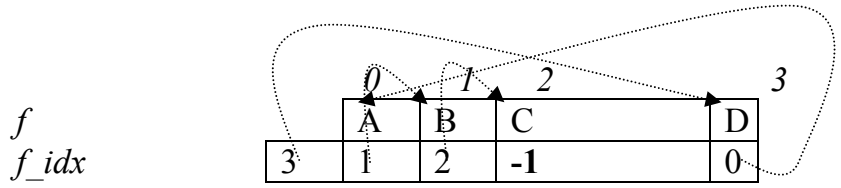
file f (chứa dữ liệu thực tế). Khi đó, *các thao tác chèn, xóa sẽ thực hiện nhanh hơn*.

Ví dụ 2: với cùng mục đích như ví dụ 1, ta dùng 2 file: file *f* để chứa các đối tượng thực sự A, B, C và file *f_idx* dùng để chứa số thứ tự bắt đầu của các đối tượng tương ứng trong *f* như sau:

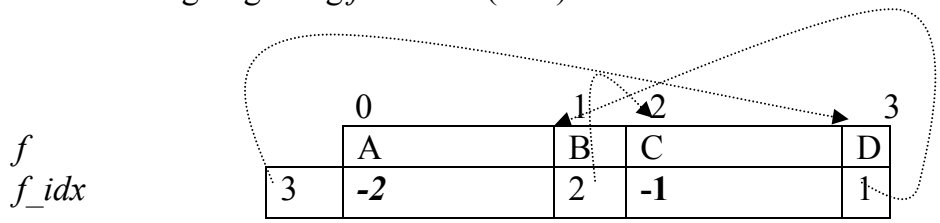


trong đó: các phần tử của *f_idx*: 0,1,2 lần lượt chỉ số thứ tự (bắt đầu) của đối tượng A, B, C trong file *f*; còn –1 (EOF) để chỉ sự kiện kết thúc file.

Việc chèn D vào *f* trước A, sẽ thực hiện như sau:



Việc xóa A, ta có thể đánh dấu xóa A (nếu cần thiết !) bằng cách gán chỉ số –2 (XOA) cho mẩu tin tương ứng trong *f_idx* và đổi lại giá trị trong file f_idx tương ứng với mẩu tin tương ứng trong *f* trước A (là D) như sau:



Tất nhiên, việc dùng kèm thêm file chỉ mục như trên có ưu điểm là làm *tăng tốc độ* thực hiện các thao tác *chèn, xóa*; ngược lại, nó sẽ *tốn thêm bộ nhớ* cho *f_idx* và cũng *làm phức tạp thêm khi viết các thao tác cơ bản* trên file, đặc biệt là các thuật toán chèn, xóa một đối tượng.

* ***Vài lưu ý khi thiết kế các thuật toán trên tập tin***:

Khi thiết kế các thuật toán trên tập tin, ngoài các *phép toán cơ bản đặc trưng cho thuật toán* (chẳng hạn: đối với các thuật toán *tìm kiếm*, ta cần để ý đến số các phép toán *so sánh;* đối với các thuật toán *sắp xếp* thì nên để ý đến số các phép toán *so sánh* và *hoán vị* hay *phép gán*; …), ta còn phải *chú ý nhiều* tới *số lần đọc và ghi đối tượng lên file*, vì thời gian cho các thao tác này chiếm thời gian khá lớn.

II. Các thao tác cơ bản trên file

Các thao tác cơ bản thường sử dụng khi làm việc trên *file chứa các đối tượng có cùng cấu trúc* là: *tạo* (xây dựng) *file*, *duyệt và khai thác file*, *tìm* hay *xóa* một *đối tượng thỏa một tính chất nào đó* của file, *chèn* (thêm) một đối tượng vào sau một đối tượng thỏa một tính chất nào đó trên file, *sửa* (thay thế) một đối tượng thỏa một tính chất nào đó trên file bởi một đối tượng khác.

II.1. Tập tin tuần tự

* Thao tác ***tạo*** file (hay nhập liệu vào file) *f:* thao tác này xây dựng file mà dữ liệu lấy từ một nguồn nào đó thông qua hàm:

Boolean *LấyĐượcMộtĐốiTượng*(ĐT)

Hàm này trả về trị *true* nếu còn lấy được một đối tượng và trả về trị *false* trong trường hợp ngược lại.

TạoFile (f)
- Bước 1: Mở file *f* để ghi mới (hay nối thêm);
- Bước 2: Trong khi còn *LấyĐượcMộtĐốiTượng*(*ĐT*)
  GhiMộtĐốiTượng(*ĐT*) vào file *f*;
- Bước 3: Đóng file *f*;

* Thao tác ***duyệt*** file (hay khai thác file) *f:* thao tác này xử lý *tất cả* các đối tượng (*mỗi đối tượng xét đúng một lần*) thỏa một tính chất *TC* nào đó của file *f*.

DuyệtFile (f, TC)
- Bước 1: Mở file *f* để đọc
- Bước 2: Trong khi còn *ĐọcĐượcMộtĐốiTượng*(ĐT) từ file *f*
  Nếu (*ĐT* có tính chất *TC*)
  thì *XửLýĐốiTượng*(*ĐT*);
- Bước 3: Đóng file *f*;

* Thao tác ***tìm*** *(tuần tự)* một đối tượng *A đầu tiên* có một tính chất *TC* nào đó trên file *f*: thao tác này trả về trị True nếu tìm thấy và False trong trường hợp ngược lại. Ngoài ra trong trường hợp tìm thấy đối tượng *A*, nó còn trả lại vị trí bắt đầu *ĐốiTượngThứ (*các mẫu tin được đánh số *bắt đầu từ 0)* của *A* trong file *f*.

Boolean Tìm (f, TC, &A, &ĐốiTượngThứ)
- Bước 1: Mở file *f* để đọc; *Thấy* ← False; *ĐốiTượngThứ* ← -1;
- Bước 2: Trong khi (CònĐọcĐượcMộtĐốiTượng *B* (từ file *f)* và Chưa *Thấy*) lặp lại các bước sau:
  . *ĐốiTượngThứ* ← *ĐốiTượngThứ* +1;
  . Nếu (*B* có tính chất *TC*) thì:
  A ← *B*; *Thấy* ← True;
- Bước 3: Đóng file *f*;

- Bước 4: Trả về trị *Thấy*;

* Thao tác ***sửa*** một đối tượng đầu tiên có một tính chất *TC* nào đó trên file *f* thành đối tượng *B (cùng kiểu với A)*: thao tác này trả về trị True nếu tìm thấy và False trong trường hợp ngược lại.

Boolean Sửa (f, TC, B)

- Bước 1: *Thấy ← Tìm(f, TC, A, Thứ);*
- Bước 2: If not(*Thấy) SửaĐược* ← False;
  Else
  Bước 2.1: Mở file *f* để ghi (và đọc);
  Bước 2.2: Nhảy đến đầu đối tượng thứ *Thứ*; Ghi *B* vào file *f*;
  Bước 2.3: Đóng file *f*;
  Bước 2.4: *SửaĐược* ←True;
- Bước 3: Trả về trị *SửaĐược*;

* Thao tác ***xóa*** một đối tượng đầu tiên có một tính chất *TC* nào đó trên file *f:* thao tác này trả về trị True nếu tìm thấy đối tượng có tính chất *TC* và False trong trường hợp ngược lại.

Boolean Xóa (f, TC)

- Bước 1: *Thấy ← Tìm(f, TC, A, Thứ);*
- Bước 2: If not(*Thấy*) *XóaĐược* ← False;
  Else
  Bước 2.1: Mở file *f* để đọc; Mở file phụ *f_phu* để ghi;
  Bước 2.2: for *(đếm←0; đếm<Thứ; đếm=đếm+1)*
  { Đọc một đối tượng *B* từ file *f;*
  Ghi đối tượng *B* lên file *f_phu*;
  }
  Bước 2.3: Đọc một đối tượng *B* từ file *f;*
  // bỏ qua đối tượng *B* có tính chất *TC*, không ghi lên file *f_phu*
  Bước 2.4: Trong khi (CònĐọcĐượcMộtĐốiTượng *B* (từ file *f)*) lặp lại bước sau:
  Ghi đối tượng *B* lên file *f_phu*;
  Bước 2.5:. Đóng file *f;* Đóng file *f_phu*;
  . Xóa file *f;* Đổi tên file *f_phu* thành *f*;
  Bước 2.6: *XóaĐược* ←True;
- Bước 3: Trả về trị *XóaĐược;*

* Thao tác ***chèn*** một đối tượng *C* sau một đối tượng đầu tiên có một tính chất *TC* nào đó trên file *f:* thao tác này trả về trị True nếu tìm thấy đối tượng có tính chất *TC* và False trong trường hợp ngược lại.

Boolean Chèn (f, C, TC)

- Bước 1: *Thấy ← Tìm (f, TC, A, Thứ);*
- Bước 2: If not(*Thấy) ChènĐược* ← False;
  Else
  Bước 2.1: Mở file *f* để đọc và ghi;
  Bước 2.2: Nhảy đến đầu đối tượng thứ (*Thứ*+1);
  Thứ ← Thứ +1;
  Bước 2.3: Trong khi (CònĐọcĐượcMộtĐốiTượng *B* (từ file *f)*) lặp lại các bước sau:
  Ghi đối tượng *C* lên file *f* tại vị trí thứ *Thứ;*
  C ← B;
  Thứ ← Thứ +1;
  Bước 2.4: Ghi đối tượng *C* lên file *f* tại vị trí thứ *Thứ*;
  Bước 2.5: Đóng file *f; ChènĐược* ←True;
- Bước 3: Trả về trị *ChènĐược*;

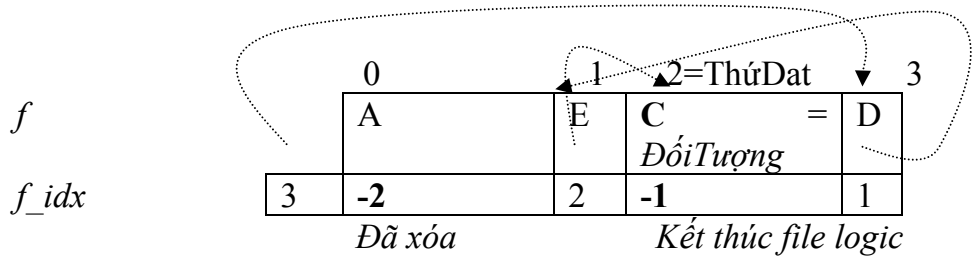
<u>Nhận xét</u>: Việc *thêm một trường đánh dấu xóa (kiểu logic)* vào kiểu của đối tượng sẽ có *ưu, nhược* điểm gì? Khi đó, *các thao tác cơ bản* trên file kiểu tuần tự sẽ *thay đổi* ra sao? Kiểm chứng lại chúng bằng chương trình (*bài tập*).

II.2. Tập tin chỉ mục

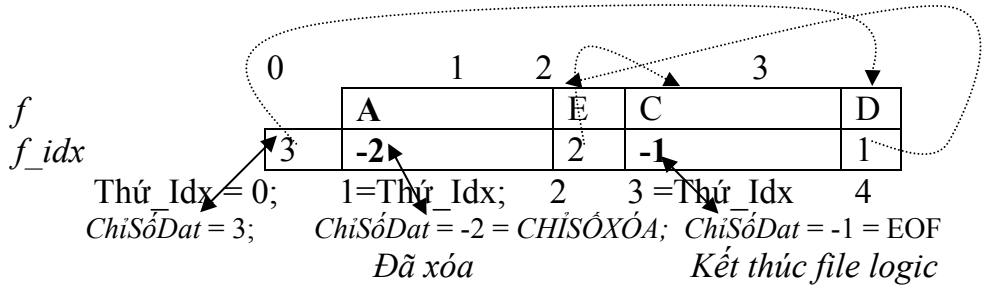
Khi làm việc với file chỉ mục, ta luôn xét file f (chứa các đối tượng thật sự) kèm với file chỉ mục f_idx tương ứng (chứa số thứ tự bắt đầu của các đối tượng tương ứng trong file f). Ký hiệu: ***F = (f, f_idx)***, *EOF = -1* là chỉ số kết thúc file, *CHỈSỐXÓA = -2* là chỉ số mẩu tin bị xóa.

Trong các thuật toán cơ bản trình bày trong phần này, ta sẽ sử dụng những thao tác sơ cấp sau đây:

- ***Đọc1ĐốiTượngTrongFileDat(f_dat, Thứ_Dat, &ĐốiTượng)***: có tác dụng đọc 1 đối tượng *ĐốiTượng* ở vị trí thứ *Thứ_Dat* từ file dữ liệu *f. Việc đọc bị thất bại nếu Thứ_Dat=EOF (hết file logic !) hoặc Thứ_Dat= CHỈSỐXÓA (mẩu tin đã bị xóa);*

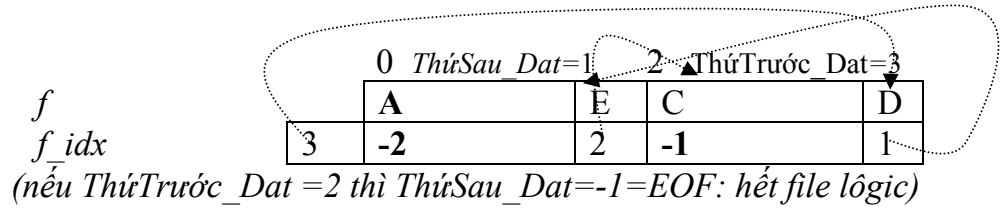


- ***Đọc1ĐốiTượngTrongFileIdx (f_idx, Thứ_Idx, &ChỉSốDat)***: có tác dụng đọc nội dung trong file chỉ mục *f_idx* tại vị trí thứ *Thứ_Idx*, cho kết quả là chỉ số *ChỉSốDat* trong file f *(nếu ChỉSốDat =EOF: hết file logic !);*



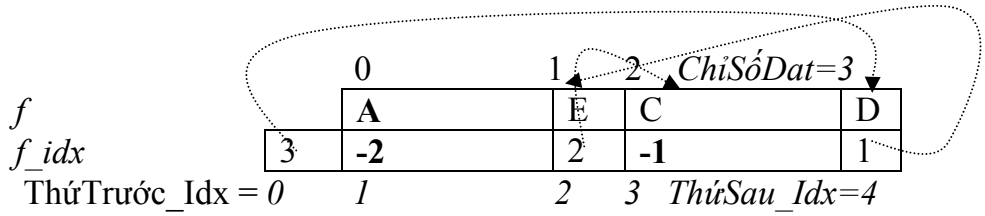
- ***Ghi_1_PTửTạiVịTrí(g, Thứ, PTử)***: có tác dụng ghi một phần tử *PTử* tại vị trí thứ *Thứ* vào file *g*.

- ***ThứSau_Dat = NextDat(f_idx, ThứTrước_Dat)***: có tác dụng trả lại số thứ tự bắt đầu *ThứSau_Dat* của mẩu tin kế tiếp (theo chỉ mục) của mẩu tin tại vị trí thứ *ThứTrước_Dat* trong file *f* (chính là nội dung của mẩu tin thứ *ThứTrước_Dat+1* trong file *f_idx) (nếu ThứSau_Dat =EOF: hết file logic !);*



(nếu ThứTrước_Dat =2 thì ThứSau_Dat=-1=EOF: hết file lôgic)

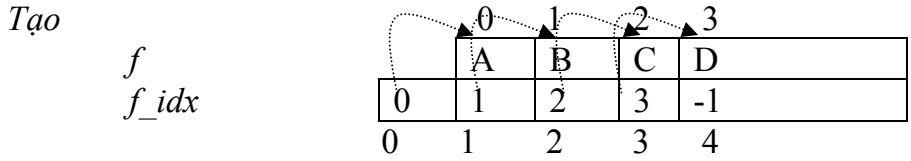
- ***ThứSau_Idx = NextIdx (f_idx, ThứTrước_Idx, &ChỉSốDat)***: có tác dụng trả lại nội dung *ChỉSốDat* của mẩu tin thứ *ThứTrước_Idx* trong file *f_idx* và số thứ tự bắt đầu *ThứSau_Idx (ThứSau_Idx* chính là *ChỉSốDat +1)* của mẩu tin kế tiếp theo thứ tự chỉ mục của mẩu tin tại vị trí thứ *ThứTrước_Idx* trong file *f_idx*.



* Thao tác ***tạo*** file (hay nhập liệu vào file) chỉ mục *F*: thao tác này xây dựng file mà dữ liệu được lấy từ một nguồn nào đó thông qua hàm:

Boolean *LấyĐượcMộtĐốiTượng*(ĐT)

Hàm này trả về: trị *true* nếu còn lấy được một đối tượng và trị *false* trong trường hợp ngược lại.



TạoFileIdx (f)

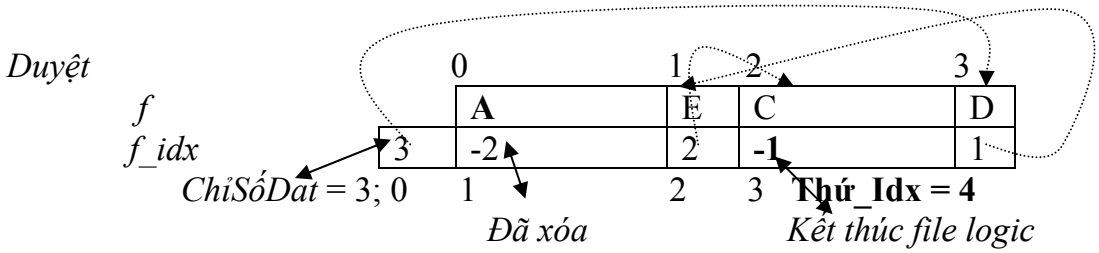
- Bước 1: Mở file *F* để ghi; *SốĐốiTượngLấyĐược* ← 0;
- Bước 2: Trong khi còn *(LấyĐượcMộtĐốiTượng*(ĐT)), lặp lại các bước sau:

  Bước 2.1: *GhiMộtĐốiTượng*(ĐT) vào cuối file *f_dat*;

  Bước 2.2: *GhiMộtSố* (*SốĐốiTượngLấyĐược*) vào cuối file *f_idx*;

  Bước 2.3: *SốĐốiTượngLấyĐược* ←*SốĐốiTượngLấyĐược +1*;
- Bước 3: *GhiMộtSố* (*EOF*) vào cuối file *f_idx*; Đóng file *F*;

* Thao tác ***duyệt*** file chỉ mục F: thao tác này xử lý tất cả các đối tượng thỏa một tính chất *TC* nào đó của file *F*.



DuyệtIdx (F, TC)

- Bước 1: Mở file *F* để đọc; *Đọc1ĐốiTượngTrongFileIdx (f_idx, 0, &ChỉSốDat);*

  // hay: Đọc 1 mẩu tin (đầu) *ChỉSốDat* của *f_idx*;
- Bước 2: Trong khi (*ChỉSốDat* ≠ *EOF*) // hay chưa hết file lôgic

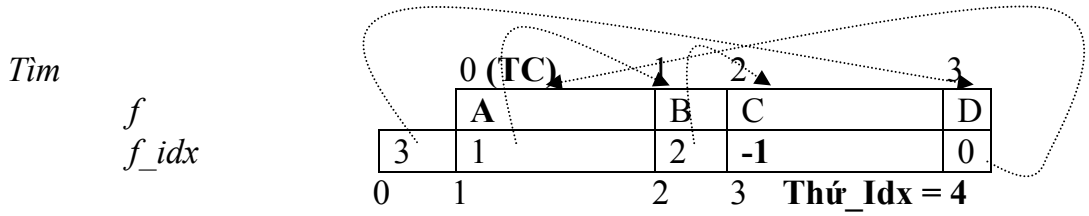
  lặp lại các bước sau:

  . *Đọc1ĐốiTượngTrongFileDat(f,ChỉSốDat,ĐốiTượng);*

  . If (*ĐốiTượng* có tính chất *TC)* XửLý(*ĐốiTượng);*

  . *ChỉSốDat = NextDat(f_idx, ChỉSốDat);*
- Bước 3: Đóng file *F*;

* Thao tác ***tìm*** *(tuần tự)* một đối tượng đầu tiên (chưa bị xóa) *A* có một tính chất *TC* nào đó trên file chỉ mục *F:* thao tác này trả về trị True nếu tìm thấy và False trong trường hợp ngược lại. Ngoài ra, trong trường hợp tìm thấy, nó còn trả lại vị trí *Thứ_Idx* của mẩu tin trong file chỉ mục *f_idx* mà nội dung của nó là vị trí bắt đầu của đối tượng tìm thấy *A*.



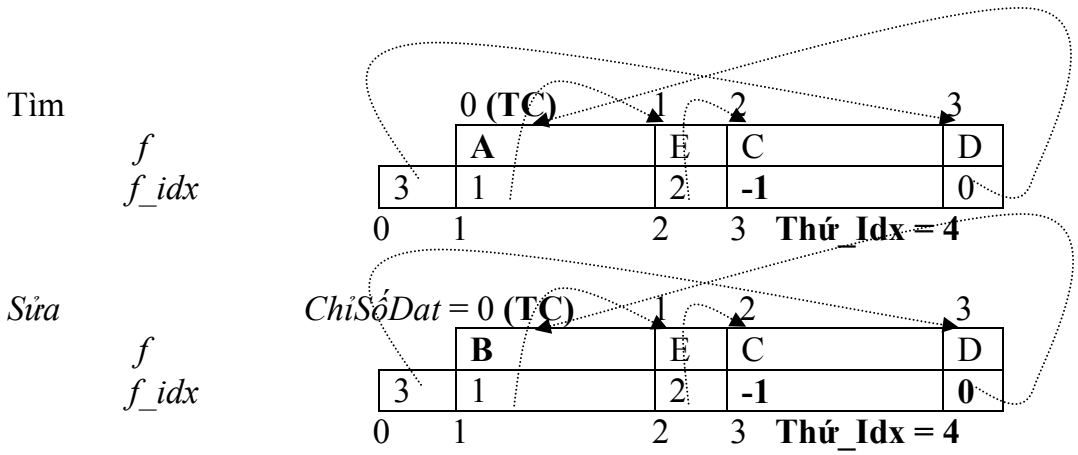
Boolean TìmIdx (F, TC, &A, &Thứ_Idx)

- Bước 1: Mở file *F* để đọc; *Thấy* ← False; *Thứ_Idx* ← 0;
  ThứSau_Idx = NextIdx (f_idx, Thứ_Idx,ChỉSốDat);
- Bước 2: Trong khi *(ChỉSốDat ≠ EOF or Chưa Thấy)* lặp lại các bước sau:

```
{   . Đọc1ĐốiTượngTrongFileDat(f,ChỉSốDat,ĐốiTượng);
    . If (ChỉSốDat == CHỈSỐXÓA)
      {. Thông báo lỗi cập nhật sai đối tượng đã bị xóa;
       . Chuyển sang bước 3;
      }
    . If (ĐốiTượng có tính chất TC)
      {. A ← ĐốiTượng;
       . Thấy ← True;
      }
    Else
      {. Thứ_Idx ← ThứSau_Idx;
       . ThứSau_Idx = NextIdx (f_idx, Thứ_Idx,ChỉSốDat);
      }
}
```

- Bước 3: Đóng file *F;*
- Bước 4: Trả về trị *Thấy*;

* Thao tác ***sửa*** một đối tượng đầu tiên (chưa bị xóa) có một tính chất *TC* nào đó trên file chỉ mục *F* thành đối tượng *B*: thao tác này trả về trị True nếu tìm thấy đối tượng cần sửa và False trong trường hợp ngược lại.



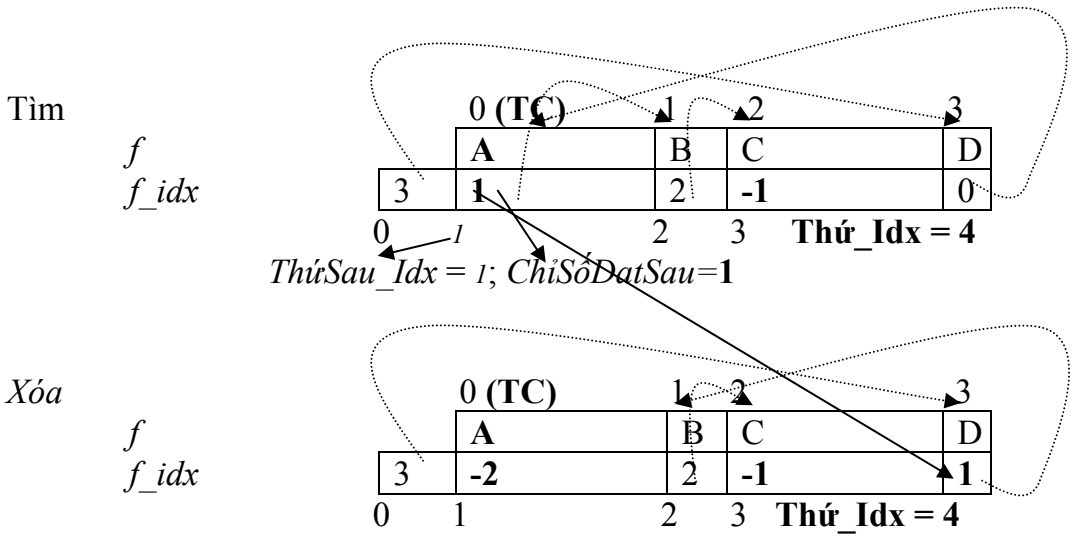
Boolean SửaIdx (F, TC, B)

- Bước 1: *Thấy ← TìmIdx (F, TC, A, Thứ_Idx);*
- Bước 2: If not(*Thấy) SửaĐược* ← False;
  Else
  Bước 2.1: Mở file *F* để ghi (và đọc);
  Bước 2.2:*ĐọcMộtĐốiTượngTrongFileIdx (f_idx,Thứ_Idx,ChỉSốDat);*
  Ghi_1_PTửTạiVịTrí(f, ChỉSốDat, B);
  Bước 2.3: Đóng file *F; SửaĐược* ←True;
- Bước 3: Trả về trị *SửaĐược;*

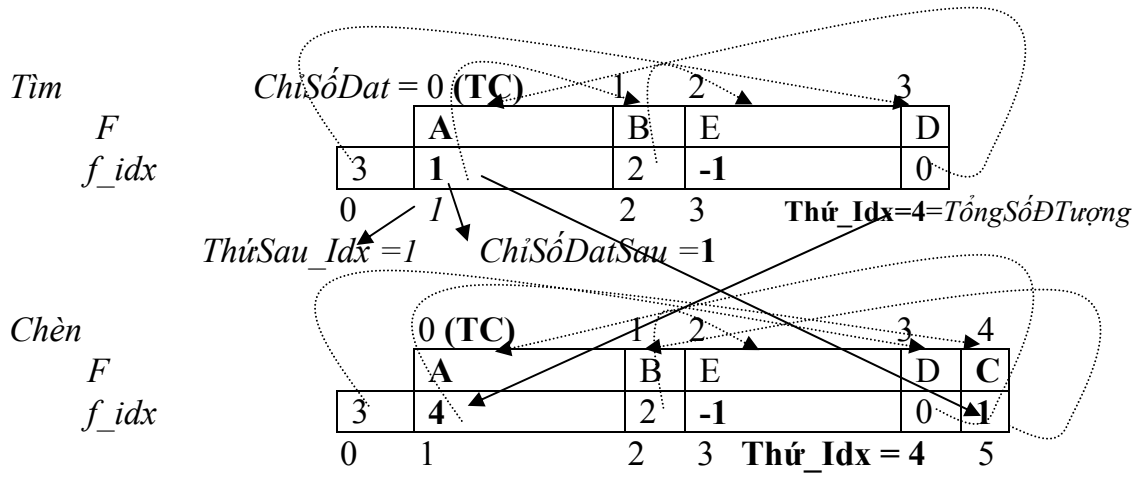
* Thao tác ***xóa*** một đối tượng đầu tiên (chưa bị xóa) có một tính chất *TC* nào đó trên file chỉ mục *F:* thao tác này trả về trị True nếu tìm thấy đối tượng cần xóa và False trong trường hợp ngược lại.

Boolean XóaIdx (F, TC)

- Bước 1: *Thấy ← TìmIdx (F, TC, A, Thứ_Idx);*
- Bước 2: If not(*Thấy) XóaĐược* ← False;
  Else
  Bước 2.1: Mở file *f_idx* để ghi và đọc;
  Bước 2.2: *ThứSau_Idx = NextIdx (f_idx, Thứ_Idx,ChỉSốDat);*
  . ĐọcMộtĐốiTượngTrongFileIdx(f_idx,ThứSau_Idx,ChiSốDatSau);
  . Ghi_1_PTửTạiVịTrí(f_idx, Thứ_Idx, ChỉSốDatSau);
  . Ghi_1_PTửTạiVịTrí(f, ThứSau_Idx, CHỈSỐXÓA);
  Bước 2.3: Đóng file *f_idx*;
  Bước 2.4: *XóaĐược* ←True;
- Bước 3: Trả về trị *XóaĐược;*



* Thao tác ***chèn*** một đối tượng *C* sau một đối tượng đầu tiên (chưa bị xóa) có một tính chất *TC* nào đó trên file chỉ mục *F*: thao tác này trả về trị True nếu tìm thấy đối tượng có tính chất *TC* và False trong trường hợp ngược lại.



Boolean ChènIdx (F, C, TC)

- Bước 1: *Thấy ← TìmIdx (F, TC, A, Thứ_Idx);*
- Bước 2: If not(*Thấy) ChènĐược* ← False;
  Else
  Bước 2.1: Mở file *F* để đọc và ghi;
  Bước 2.2:.Tìm *TổngSốĐốiTượng* trong file f *(trước khi chèn)*;
  . Ghi *C* vào cuối *f*;
  . *ThứSau_Idx = NextIdx (f_idx, Thứ_Idx,ChỉSốDat);*
  . *ĐọcIĐốiTượngTrongFileIdx(f_idx,ThứSau_Idx,ChỉSốDatSau);*
  . Ghi *ChỉSốDatSau* vào cuối file *f_idx*;
  . *Ghi_1_PTửTạiVịTrí(f_idx, ThứSau_Idx, TổngSốĐốiTượng);*
  Bước 2.3: Đóng file *F; ChènĐược* ←True;
- Bước 3: Trả về trị *ChènĐược;*

* <u>*Nhận xét*</u>:

- Lưu ý rằng, trong các thuật toán trên, ta *không cập nhật lại những mẩu tin (đối tượng) đã bị xóa*. Nếu muốn truy cập hoặc phục hồi lại các mẩu tin này thì cần viết thêm các thủ tục tương ứng (*bài tập*).
- Với kiểu file chỉ mục để chứa dãy các đối tượng có cùng cấu trúc, ta còn có thể *tổ chức theo cách khác*, trong đó *mỗi phần tử trong file là một cấu trúc gồm hai trường: dữ liệu Data và địa chỉ nút dữ liệu tiếp theo Next* (tương tự như cách tổ chức danh sách liên kết ở bộ nhớ trong). Cách tổ chức này tuy *đơn giản* nhưng sẽ *gặp bất tiện trong nhiều thao tác phức tạp hơn*, chẳng hạn khi ta muốn *"**sắp xếp động**"* cùng một dãy các đối tượng dữ liệu *theo nhiều (chưa biết trước số lượng) quan hệ thứ tự khác nhau* trên các trường khác nhau của dữ liệu. Tình huống đó sẽ *dễ dàng thực hiện bằng cách tổ chức thêm nhiều file chỉ mục f_idx* như trên với file dữ gốc f không đổi.

- Khi tổ chức file *f* theo kiểu chỉ mục như trên, tuy phải dùng thêm bộ nhớ phụ cho file *f_idx*, nhưng kiểu của mỗi mẩu tin của nó chỉ là kiểu nguyên, nên nếu kích thước của mỗi đối tượng của file *f* khá lớn (thường gặp trong thực tế) thì *dung lượng bộ nhớ phụ cho file f_idx là không đáng kể* !
- Bù lại, nếu phải *dùng nhiều phép chèn, xóa* các đối tượng trên file *f (trên thực tế thường xảy ra)*, thì *thời gian thực hiện sẽ nhanh (không phụ thuộc vào kích thước mẩu tin)*. Ngoài ra, khi cần viết các thuật toán phức tạp trên tập tin, chẳng hạn sắp xếp, thì *thời gian đáng kể để thực hiện cho các thuật toán này là để sao chép các đối tượng từ tập tin này sang tập tin khác*. Nếu tổ chức file theo kiểu chỉ mục, thì chỉ phải sao chép các kiểu dữ liệu nguyên (file chứa dữ liệu thật sự không đổi !). Khi đó thời gian cho các thuật toán này (thường cần dùng nhiều file phụ) sẽ rút ngắn đáng kể và bộ nhớ cần dùng cho các file phụ (chỉ cần dùng thêm file chỉ mục dạng *idx*) sẽ giảm đáng kể !
- Trong giáo trình "*Cấu trúc dữ liệu và giải thuật 1*", ta đã biết khi dữ liệu được lưu trữ ở *bộ nhớ trong*, việc dùng *danh sách liên kết* sẽ thực hiện nhanh các thao tác chèn, xóa dữ liệu so với kiểu mảng, nhưng việc tìm kiếm (tuyến tính) trên chúng vẫn chậm; để vượt qua tất cả các hạn chế này (chèn, xóa cũng như sắp xếp, tìm kiếm), ta có thể dùng *các cấu trúc dữ liệu "nhiều chiều"* hơn danh sách liên kết tuyến tính như *cây tìm kiếm nhị phân BST, cây cân bằng AVL*. Tương tự như vậy, khi dữ liệu được lưu trữ ở *bộ nhớ ngoài*, tuy *file chỉ mục* thực hiện tốt hơn các thao tác chèn, xóa so với file tuần tự, nhưng việc *tìm kiếm dữ liệu trên chúng vẫn thực hiện rất chậm theo cách tuyến tính*. Để khắc phục cả hạn chế này, ta cần *tổ chức file theo kiểu cây BST, cây cân bằng AVL* hay *B-cây* sẽ trình bày trong chương tiếp theo (*bài tập**).

III. <u>Sắp xếp trên file</u>

Giả sử ta cần sắp (tăng) các đối tượng có cùng kiểu *T* trong file *f* cho trước, với điều kiện là trong kiểu T có một trường (gọi là trường *khóa key*) mà trên miền trị của trường đó có một *quan hệ thứ tự* $\propto$ cho trước (một quan hệ hai ngôi có các tính chất: phản xạ, phản xứng và bắc cầu; ta thường gặp quan hệ $\propto$ là quan hệ $\leq$ thông thường).

* <u>*Định nghĩa 1*</u>*: (đường chạy với chiều dài cố định)*

Một *đường chạy* (theo trường khóa *key*) có *chiều dài cố định k* là một dãy gồm *k* đối tượng $d_1, d_2, \ldots, d_k$ được sắp theo một trường khóa *key*:

$$d_1.key \propto d_2.key \propto \ldots \propto d_k.key$$

* *Định nghĩa 2: (đường chạy tự nhiên với chiều dài không cố định)*

Một *đường chạy (tự nhiên) r* (theo trường khóa *key*) trên file f là một dãy con gồm các đối tượng r = {d_m, d_{m+1}, …,d_n } thỏa các tính chất sau:

$$\begin{cases} d_i.key \le d_{i+1}.key, \forall\, i \in [m,n) \cap N,\ n \ge m \\ d_{m-1}.key > d_m.key, \quad \text{nếu } m>1 \\ d_n.key > d_{n+1}.key, \quad \text{nếu } \exists\, d_{n+1} \end{cases}$$

* *Định nghĩa 3: (thao tác trộn)*

Trộn 2 đường chạy *r1, r2* có chiều dài lần lượt *d1* và *d2* là tạo ra đường chạy mới *r* (gồm tất cả các đối tượng từ *r1* và *r2*) có chiều dài *d1+d2*.

III.1. Trộn trực tiếp *(Straight Merge)*

* *Ý tưởng phương pháp:* Sử dụng thêm *2 file phụ f1 và f2* để thực hiện các phép ***phân phối*** *luân phiên* các đường chạy có chiều dài *k* là lũy thừa của 2 của *f* vào *f1* và *f2*. Sau đó ***trộn*** *luân phiên* các đường chạy có chiều dài *k* từ *f1* và *f2* thành các đường chạy dài gấp đôi *2*k* vào *f*. Gấp đôi chiều dài đường chạy *k* ← *2*k*. Lặp lại các phép phân phối và trộn luân phiên các đường chạy như trên cho đến khi chiều dài đường chạy *k* ≥ số phần tử của file *f (*trong *f chỉ còn lại một đường chạy* !) thì các phần tử trong f được sắp.

* *Thuật toán*:

SắpXếpTrộnTrựcTiếp(&f)

- Bước 1: *DàiĐườngChạy ← 1;*
- Bước 2: Lặp lại các bước sau:
  - Bước 2.1: Gọi thuật toán "*PhânPhốiTrựcTiếp*" để phân phối lần lượt các đường chạy có chiều dài *DàiĐườngChạy* từ *f* vào *f[1]* và *f[2];*
  - Bước 2.2: Gọi thuật toán "*TrộnTrựcTiếp"* để trộn lần lượt các đường chạy có chiều dài *DàiĐườngChạy* tương ứng trong *f[1]* và *f[2]* vào *f.*
  - Bước 2.3: *DàiĐườngChạy ← 2 * DàiĐườngChạy;*

  Cho đến khi (*DàiĐườngChạy* >= số phần tử của file *f);*

+ ***PhânPhốiTrựcTiếp(f, &f1, &f2, DàiĐườngChạy)***

- Bước 1:. Mở file *f[1]* và *f[2]* để ghi, mở file *f* để đọc;
  . *FileThứ*←1;*PTửThứ*←0;
- Bước 2: Trong khi (chưa kết thúc *f*) lặp lại các bước sau:
  { . *PTửThứ ← PTửThứ +1*;
  . Sao một phần tử của *f* vào *f[FileThứ]*;
  . If (*PTửThứ == DàiĐườngChạy)*

{ . *PTửThứ*←0;
. If (*FileThứ* < *2) FileThứ* ← *FileThứ+1;*
else *FileThứ* ← *1;*
}
}

- Bước 3: Đóng các file *f*, *f[1]* và *f[2]*.

+ ***TrộnTrựcTiếp(f[1], f[2], &f, DàiĐườngChạy)***

- Bước 1:. *SốPTửCầnChépVàoFilef* ← *SốPTửCủaFilef;*
. Mở file *f[1]* và *f[2]* để đọc, mở file *f* để ghi;
. *Đọc1ĐTượng x1* của *f[1]; Đọc1ĐTượng x2* của *f[2];*
// gọi *r[i]* là chiều dài đường chạy của *f[i], i=1,2*
- Bước 2: Lặp lại các bước sau:
. If *(DàiĐườngChạy* ≤ *SốPTửCầnChépVàoFilef) r[1]* ← *DàiĐườngChạy;*
else *r[1]* ← *SốPTửCầnChépVàoFilef;*
. *SốPTửCầnChépVàoFilef* ← *SốPTửCầnChépVàoFilef – r[1];*
. If *(DàiĐườngChạy* ≤ *SốPTửCầnChépVàoFilef) r[2]* ← *DàiĐườngChạy;*
else *r[2]* ← *SốPTửCầnChépVàoFilef;*
. *SốPTửCầnChépVàoFilef* ← *SốPTửCầnChépVàoFilef – r[2];*
. Trong khi (*r[1]*>0 và *r[2]*>0) thực hiện:// *chưa hết 2 đường chạy*
If (x1<x2)
{ GhiĐTượng *x1* vào *f*; *r[1]* ← *r[1]-1;*
If (chưa kết thúc file *f[1]*) *Đọc1ĐTượng x1* của *f[1];*
}
else
{ GhiĐTượng *x2* vào *f*; *r[2]* ← *r[2]-1;*
If (chưa kết thúc file *f[2]*) *Đọc1ĐTượng x2* của *f[2];*
}
. Trong khi (*r[1]*>0) thực hiện:// *f[1] chưa hết đường chạy*
{ GhiĐTượng *x1* vào *f*; *r[1]* ← *r[1]-1;*
If (chưa kết thúc file *f[1]*) *Đọc1ĐTượng x1* của *f[1];*
}
. Trong khi (*r[2]*>0) thực hiện:// *f[2] chưa hết đường chạy*
{ GhiĐTượng *x2* vào *f*; *r[2]* ← *r[2]-1;*
If (chưa kết thúc file *f[2]*) *Đọc1ĐTượng x2* của *f[2];*
}
Cho đến khi (SốPTửCầnChépVàoFilef==0);
- Bước 3: Đóng các file *f*, *f[1]* và *f[2]*.

Ví dụ 3: giả sử ta cần sắp tăng file *f* sau:
f : 2, 1, 4, 5, 7

- *DàiĐườngChạy = 1:*
  Phân phối f thành*:*
    f1: 2; 4; 7
    f2: 1; 5
  Trộn *f1* và *f2* vào *f* thành các đường chạy có chiều dài *2*:
    f : 1, 2; 4, 5; 7
- *DàiĐườngChạy = 2:*
  Phân phối f thành*:*
    f1: 1, 2; 7
    f2: 4, 5
  Trộn *f1* và *f2* vào *f* thành các đường chạy có chiều dài *4*:
    f : 1, 2, 4, 5; 7
- *DàiĐườngChạy = 4:*
  Phân phối f thành*:*
    f1: 1, 2, 4, 5
    f2: 7
  Trộn *f1* và *f2* vào *f*:
    f : 1, 2, 4, 5, 7
- *DàiĐườngChạy = 8 (>5):* dừng *!*

* *Nhận xét*:

- Với phương pháp trộn trực tiếp, số lần *phân phối và trộn* khoảng: $k=\log_2(n)$. Do mỗi lần phân phối hoặc trộn, ta cần n lần sao chép các đối tượng từ tập tin này sang tập tin khác, nên tổng số các đối tượng cần sao chép trong trường hợp xấu nhất là:

$$T_{xấu}(n) = 2 * n * \log_2(n)$$

- Trong giải thuật trộn trên đây, để đơn giản cho việc trình bày, ta chỉ sử dụng 2 file phụ *f1* và *f2* trong các giai đoạn phân phối và trộn. Thật ra, dựa vào các ý tưởng cơ bản trên đây, ta có thể mở rộng thuật toán khi sử dụng đồng thời nhiều hơn 2 tập tin phụ *f1, f2, ...,ft (t>2)* để thực hiện các giai đoạn phân phối và trộn với độ dài các đường chạy k là lũy thừa của t. Khi đó, tổng số các đối tượng cần sao chép trong trường hợp xấu nhất là:

$$T_{xấu}(n) = 2 * n * \log_t(n)$$

- Qua ví dụ trên, ta thấy phương pháp trộn trực tiếp có nhược điểm sau: do luôn sử dụng chiều dài đường chạy cố định *k* tại mỗi vòng lặp *phân phối và trộn* nên *không tận dụng được tình trạng* "*tốt tự nhiên*" của dữ liệu (trong ví dụ trên, đáng lẽ ta có thể dừng ngay khi vừa thực hiện xong bước lặp với *DàiĐườngChạy = 2*; lúc đó: 1,2,4,5,7 là một *đường*

chạy tự nhiên !). Vì lẽ đó, ta có thể cải tiến phương pháp trộn trực tiếp thành phương pháp trộn tự nhiên sau đây.

III.2. Trộn tự nhiên *(Natural Merge)*

* *Ý tưởng phương pháp:* Sử dụng thêm *2 file phụ f1 và f2* để thực hiện các phép ***phân phối*** *luân phiên* các *đường chạy tự nhiên* của *f* vào *f1* và *f2*. Sau đó ***trộn*** *luân phiên* các *đường chạy tự nhiên* từ *f1* và *f2* thành các đường chạy dài hơn vào *f*. Lặp lại các phép phân phối và trộn luân phiên các *đường chạy tự nhiên* như trên cho đến khi trong *f chỉ còn lại một đường chạy* thì các phần tử trong f được sắp.

* *Thuật toán*:

SắpXếpTrộnTựNhiên (f)

Lặp lại các bước sau:

Bước 1: Gọi thuật toán "*PhânPhốiTự Nhiên(f,f1,f2)"* để phân phối các đường chạy (tự nhiên) trong *f* lần lượt vào *f1* và *f2;*

Bước 2:. Gán *SốĐườngChạy* ← *0;*

. Gọi thuật toán *"TrộnTựNhiên (f1,f2,f,SốĐChạy)"* để trộn các đường chạy (tự nhiên) tương ứng trong *f1* và *f2* vào *f;*

Cho đến khi *SốĐườngChạy = 1.*

PhânPhốiTựNhiên(f,f1,f2)

/* Thuật toán *phân phối luân phiên* các *đường chạy tự nhiên* của *f* vào *f1* và *f2* */

- Bước 1: Mở file *f1* và *f2* để ghi; mở file *f* để đọc;
- Bước 2: Trong khi (chưa kết thúc *f*) lặp lại các bước sau:

  Bước 2.1: Sao một đường chạy (tự nhiên) từ *f* vào *f1* (lặp lại việc đọc một *ĐốiTượng_1* của *f* và ghi nó vào *f1* cho đến khi *ĐốiTượng_2* tiếp theo trong *f* nhỏ hơn *ĐốiTượng_1* vừa được sao hoặc gặp kết thúc file *f*);

  Bước 2.2: Nếu chưa đạt đến kết thúc *f*, sao một đường chạy (tự nhiên) tiếp theo của *f* vào *f2;*
- Bước 3: Đóng các file *f, f1, f2.*

TrộnTựNhiên(f1,f2,f, &SốĐChạy)

/* Thuật toán trộn các *đường chạy tự nhiên* tương ứng trong *f1* và *f2* vào *f*. *SốĐườngChạy* là số các *đường chạy tự nhiên* tạo ra trong *f* */

- Bước 1:. Mở file *f1* và *f2* để đọc; mở file *f* để ghi;

  . Khởi động *SốĐườngChạy=0;*
- Bước 2: Trong khi (chưa kết thúc *f1* và chưa kết thúc *f2*) lặp lại các bước sau:

Bước 2.1: Trong khi chưa hết đường chạy trong *f1* và chưa hết đường chạy trong *f2* làm các bước sau:

Nếu *ĐốiTượng_1* tiếp theo trong *f1* nhỏ hơn *ĐốiTượng_2* tiếp theo trong *f2* thì chép *ĐốiTượng_1* vào *f;* nếu ngược lại chép *ĐốiTượng_2* vào f;

Bước 2.2: Nếu hết đường chạy trong *f1*, sao phần còn lại của đường chạy tương ứng trong *f2* vào *f;* nếu ngược lại, sao phần còn lại của đường chạy tương ứng trong *f1* vào f;

Bước 2.3: Tăng *SốĐườngChạy* thêm 1;

- Bước 3: Sao tất cả các đường chạy còn lại trong *f1* hoặc *f2* vào *f;* ứng với mỗi đường chạy tăng *SốĐườngChạy* lên 1*;*
- Bước 4: Đóng các file *f, f1, f2.*

+ ***SaoMộtĐườngChạy(f_Nguon,&f_Đích)***

// sao một đường chạy từ *f_Nguồn* đã mở để đọc *đến f_Đích* đã mở để ghi

- Lặp lại bước sau:. *KếtThúcĐườngChạy ← False;*
  . *SaoMộtĐốiTượng(f_Nguon,f_Đích, KếtThúcĐườngChạy)*

  Cho đến khi (*KếtThúcĐườngChạy*);

+ ***SaoMộtĐốiTượng(f_Nguon, &f_Đích, &KếtThúcĐườngChạy)***

/*sao một đối tượng từ *f_Nguồn* vào *f_Đích* và cho biết đã *KếtThúcĐườngChạy* trong *f_Nguồn* hay chưa */

- Bước 1: Đọc một *ĐốiTượng HTại* từ file *f_Nguồn và ghi vào* file *f_Đích;*
- Bước 2: If (KếtThúcFile(*f_Nguồn)*) *KếtThúcĐườngChạy ← True;*
  Else {. *XemĐốiTượngTiếpTheo Sau* của file *f_Nguồn;*
  . If *(Sau < HTại) KếtThúcĐườngChạy ← True;*
  else *KếtThúcĐườngChạy ← False;*
  }

* *Ví dụ:* Sắp xếp tăng dần bằng phương pháp trộn tự nhiên tập tin f có nội dung như sau:

f: 75 55 15 20 85 30 35 10 60 40 50 25 45 80 70 65

- Phân phối (Tách hay Chia) (lần 1):
  f1: 75 15 20 85 10 60 25 45 80 65
  f2: 55 30 35 40 50 70

Trộn:

f: 55 75 15 20 30 35 40 50 70 85 10 60 25 45 80 65

- Phân phối (lần 2):
  f1: 55 75 10 60 65
  f2: 15 20 30 35 40 50 70 85 25 45 80

Trộn:

f: 15 20 30 35 40 50 55 70 75 85 10 25 45 60 65 80

- Phân phối (lần 3):
  f1: 15 20 30 35 40 50 55 70 75 85
  f2: 10 25 45 60 65 80
  Trộn:
  f: 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85

III.3. Trộn nhiều đường cân bằng *(Balanced Multiway Merge)*

Trong các giải thuật trộn trực tiếp và tự nhiên trên đây, *thời gian thực hiện chủ yếu dựa vào số lần duyệt tập tin để phân phối và trộn các đường chạy*, do mỗi lần duyệt tập tin thì toàn bộ các đối tượng của tập tin sẽ được sao chép lại. Ta có thể cải tiến chúng nhờ *giảm số lần duyệt tập tin*, bằng cách *chủ yếu chỉ dùng các quá trình trộn* mà *hạn chế dùng quá trình phân phối các đường chạy*.

Trong phương pháp trộn nhiều đường cân bằng để sắp xếp các đối tượng của tập tin *f[0]*, ta dùng thêm *N (N* chẵn) tập tin phụ *f[1], f[2], ..., f[N]*. Gọi *nh=N/2*.

TrộnNhiềuĐường CânBằng(f[0])

- *Bước 1*: Phân phối luân phiên các đường chạy (tự nhiên) từ *f[0]* lần lượt vào các tập tin phụ *f[1], f[2], ..., f[nh];*
- *Bước 2*: Lặp lại các bước sau:
  Bước 2.1: Trộn các đường chạy (tự nhiên) của *f[1], f[2], ..., f[nh] và* lần lượt luân phiên phân phối vào các tập tin *f[nh+1], f[nh+2], ..., f[N];*
  Bước 2.2: Nếu số đường chạy (tự nhiên) sau khi trộn lớn hơn 1 thì trộn các đường chạy của *f[nh+1], f[nh+2], ..., f[N]* và lần lượt luân phiên phân phối vào các tập tin *f[1], f[2], ..., f[nh];*

  Cho đến khi số đường chạy (tự nhiên) sau khi trộn bằng 1;

* *Nhận xét*: Với phương pháp trộn N đường cân bằng, số lần duyệt tập tin là: $k=\log_{nh}(n)$. Do mỗi lần duyệt tập tin, ta cần n lần sao chép, nên tổng số các đối tượng cần sao chép trong trường hợp xấu nhất là:

$$T_{xấu}(n) = n * \log_{nh}(n), \qquad \text{với } nh = N/2$$

* *Ví dụ:* Sắp xếp tăng dần bằng phương pháp trộn với N = 4 đường cân bằng cho tập tin *f* có nội dung như sau:

f: 75 55 15 20 85 30 35 10 60 40 50 25 45 80 70 65

- *Bước 1*: (Phân phối f vào 2 file f1 và f2)

f1: 75 15 20 85 10 60 25 45 80 65

f2: 55 30 35 40 50 70

- *Bước 2*: Lặp

. *Lần 1*:

Trộn luân phiên các đường chạy (tự nhiên) từ f1, f2 lần lượt vào f3 và f4:

f3: 55 75 10 60 65

f4: 15 20 30 35 40 50 70 85 25 45 80

Trộn luân phiên các đường chạy (tự nhiên) từ f3, f4 lần lượt vào f1 và f2:

f1: 15 20 30 35 40 50 55 70 75 85

f2: 10 25 45 60 65 80

. *Lần 2*:

Trộn luân phiên các đường chạy (tự nhiên) từ f1, f2 lần lượt vào f3 và f4:

f3: 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85

f4: trống.

Chỉ còn 1 đường chạy: f3 đã được sắp, xóa f và *đổi tên f3 thành f.*

Chương 2

B - CÂY

Cấu trúc dữ liệu *cây nhị phân hay cây BST được dùng cho các sơ đồ tìm kiếm nội trú*: nghĩa là khối dữ liệu cần tìm phải đủ nhỏ để có thể lưu trữ toàn bộ vào bộ nhớ trong.

Để giải quyết cho *sơ đồ tìm kiếm ngoại trú, với khối lượng dữ liệu lớn, phải lưu trữ trên bộ nhớ ngoài, người ta nghiên cứu cây chứa những nút có nhiều nhánh gọi là cây nhiều nhánh.*

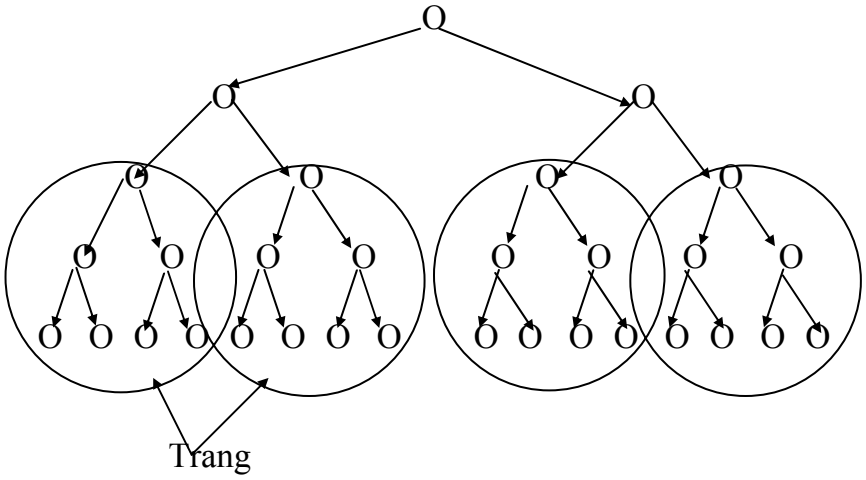
I. Đặc điểm cây nhiều nhánh

- *Các nút của cây được lưu trữ trên bộ nhớ ngoài.*
- *Các con trỏ được biểu diễn bởi địa chỉ bộ nhớ ngoài (thay vì địa chỉ bộ nhớ trong).*
- *Nếu có một phần tử trên bộ nhớ ngoài được truy xuất thì toàn bộ một nhóm nào đó các phần tử có liên quan cũng được truy xuất theo mà không tốn nhiều thời gian.*

Điều này dẫn đến một cây được chia thành nhiều cây con và các *cây con được biểu diễn thành các đơn vị mà các phần tử của đơn vị được truy xuất đồng thời*. Ta gọi các cây con này là *các trang*.

* <u>*Ví dụ*</u>:

Cây nhị phân được chia thành nhiều trang. Mỗi trang gồm 7 nút. Những nút thuộc cùng một trang được truy xuất đồng thời.



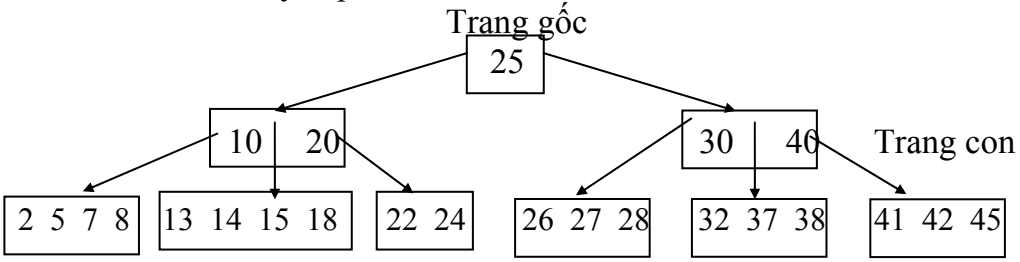
Một trong những cây nhiều nhánh quan trọng là: cấu trúc dữ liệu B - cây bậc n.

II. Định nghĩa B - cây (bậc n)

B-cây bậc n là một cây tổng quát thỏa các tính chất sau:

- Mỗi trang có ***tối đa 2* n*** **phần tử** (khóa).
- Mỗi trang (ngoại trừ trang gốc) có ***ít nhất n*** **phần tử**.
- Mỗi trang hoặc là **trang *lá***, hoặc **có *m+1 trang con*** *với m là số khóa* của trang này.
- *Mọi trang* ***lá*** phải có ***cùng mức***.
- Các khóa trên mỗi trang được ***sắp thứ tự*** *để phân các khoá lưu trữ trong các trang con* (nếu có)

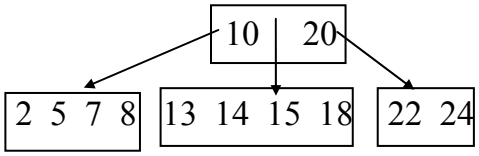
* *Ví dụ*: B - cây cấp 2 , có 3 mức:



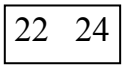
Để đơn giản trong trình bày, ta *giả sử B-cây được lưu hoàn toàn ở bộ nhớ trong*.

III. Tìm kiếm một phần tử trên B - cây

* *Ví dụ2*: Xét B - cây cấp 2 lưu trữ các số nguyên trong ví dụ trên đây. Giả sử ta cần tìm phần tử 22 có trong cây hay không ? So sánh với nút gốc ta thấy 22 < 25, ta tìm 22 trong nhánh con bên trái của nút 25:

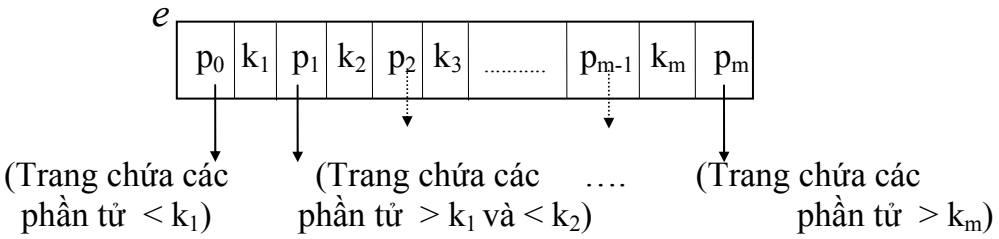


Ta lại so sánh 22 với các phần tử trong nút gốc của cây mới, ta thấy 22 > 20, ta sẽ tìm trong nhánh con phải của phần tử 20:



Trong việc ứng dụng cài đặt lưu trữ B-cây, mỗi trang sẽ được lưu trữ trên bộ nhớ thứ cấp như một khối file (một lượng thông tin lớn nhất có thể nạp vào bộ nhớ trong ở một lần truy nhập). Mỗi khi cần tìm kiếm, một trang nào đó sẽ được nạp vào bộ nhớ trong, phần còn lại của cây vẫn nằm lại trên bộ nhớ thứ cấp.

* Tổng quát, xét một trang *e* có dạng sau và cần tìm khoá x



Mỗi trang trên được nạp vào bộ nhớ trong. Trước hết, tìm *x* trong dãy khoá $k_1, k_2,..., k_m$ (Nếu m lớn thì tìm theo phương pháp nhị phân). Nếu ***thấy***, việc *tìm kết thúc*. Nếu không tìm thấy ($x \neq k_i, \forall i=1..m$) thì ta sẽ gặp một trong các tình huống sau:

+ $k_i < x < k_{i+1}$ với $1 \le i \le m-1$: tiếp tục tìm trên trang có địa chỉ do con trỏ p_i chỉ đến.
+ $k_m < x$: tiếp tục tìm trên trang do p_m trỏ đến.
+ $x < k_1$: tiếp tục tìm trên trang do p_0 trỏ đến.

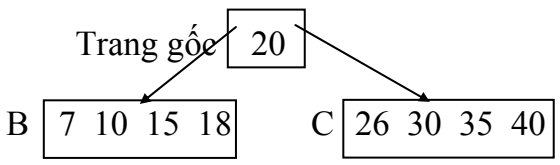
Trong trường hợp nếu có con trỏ p_i nào đó chỉ đến NULL, nghĩa là không còn trang con nữa thì trong B - cây *không có phần tử có khoá x* (***không thấy***), việc *tìm kiếm chấm dứt*.

Định nghĩa cấu trúc trang

```
const  n  =...;                  // cấp của B-cây
 #define  nn  2*n                // kích thước tối đa của một trang
typedef  …   KeyType;
typedef   Page  *Ref ;           // địa chỉ của trang
 struct  Item  { KeyType  Key;// khóa
                   Ref  p;       // chứa địa chỉ đến trang con bên phải của khóa
                   int Count;    // đại diện cho các dữ  liệu còn lại
                 };              // phần tử của trang
struct  Page   {int   m;         // số phần tử của trang
                   Ref  p0;      // địa chỉ trang con có các khoá < e1.key
                   Item  e[nn];  // các phần tử  của trang
                 };              // trang
```

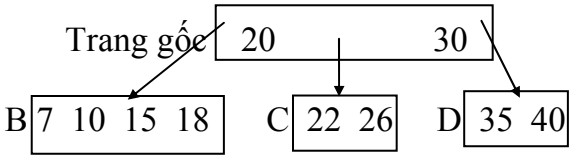
IV. Thêm một phần tử vào B - cây

* *Ví dụ*: Xét B - cây cấp 2 như sau và ta cần thêm phần tử 22:



Ta nhận thấy:

- không có khoá 22 trong cây
- không thể thêm 22 vào trang C vì nó đã đầy
- khi đó trang C được tách thành 2 trang (1 trang D mới được cấp phát)
- 2*n+1 khoá (kể cả khóa mới) trên trang C cũ được phân bố đều lên C và D mới, khoá giữa được chuyển lên một mức ở trang cha A.



(Cây sau khi thêm phần tử 22)

Quá trình tìm kiếm và thêm một phần tử vào trên B - cây

Quá trình *thêm một phần tử x vào B - cây* xảy ra như sau: Tìm *x* trên B-cây.

- Nếu tìm thấy, không cần thêm *x* vào B-cây (hoặc tăng số lần xuất hiện của x lên 1 đơn vị).
- Nếu không tìm thấy *x*, thì sẽ biết địa chỉ trang lá *L* cần thêm *x* vào. Giả sử *trang lá L* có *m* phần tử:
  - nếu *m < 2n (*trang lá *L chưa đầy)* thì việc thêm *x* chỉ xảy ra trên trang lá đó.
  - nếu *m=2n (*trang lá *L đầy)* thì phải cấp phát thêm trang lá mới L2. Phân phối đều *2*n+1* phần tử (sắp tăng *2*n* phần tử trên *L* và kể cả *x*) trên *L cũ* vào *L, L2 mới và trang cha của L* như sau: *n phần tử nhỏ nhất* của *L* cũ vẫn *giữ lại trên trang L* mới, đưa *n phần tử lớn nhất* của *L* cũ *vào* trang lá mới *L2*, còn *phần tử giữa* – phần tử thứ *n+1* trên L cũ – được *đưa vào trang cha* và chỉnh lại các địa chỉ trên trang cha chỉ đến các trang con cho phù hợp. Khi đó, nếu trang cha mới có số phần tử:

    . không lớn hơn *2*n (trang cha chưa tràn)*: thì việc thêm *x* vào B-cây kết thúc.

    . bằng *2*n +1* (*trang cha bị tràn*): thì trang cha bị tách thành 2 trang mới và làm tương tự như trên. Trong trường hợp tồi nhất, việc *tách trang* có thể *lan truyền đến trang gốc*, làm cho trang gốc tách thành hai trang, trang gốc mới được cấp phát và *B - cây sẽ tăng độ cao.* Do sự kiện này người ta nói B - cây có qui luật tăng trưởng từ lá cho đến gốc.

IV.1. <u>Giải thuật tìm và thêm một phần tử vào B - cây</u>

. *<u>Input</u>*: - x: khoá cần tìm và thêm vào B-cây
- root: địa chỉ trang gốc

. *<u>Output</u>*: - Nếu việc thêm thành công, trả về trị đúng 1: nếu có sự tách trang và dẫn đến chiều cao của cây tăng thì:
. *h* = 1 (true)
. *u* là phần tử được thêm vào trang cha của *a*.
- Trả về trị sai 0 trong trường hợp ngược lại.

Int Search_Insert (KeyType x, Ref a, Boolean &h, Item &u)

```
{   if (a == NULL) // x không có trên cây
        { h = 1; u.Key = x; u.p = NULL;
        }
    else
        { // tìm khóa x trên trang a
                "Tìm nhị phân x trên trang a";
                if  "Tìm thấy"
                {       "Tăng số lần xuất hiện khoá x lên 1";
                        h = 0;
                 }
                else
                {   Search (x,TrangCon(a),h,u);
                    if (h)      // chuyển u lên cây
                        if (Số phần tử trên trang a < 2n)
                        {       Chèn u vào trang a;
                                h = 0;
                        }
                        else if (!"Tách trang và chuyển phần tử giữa lên") return 0;
                }
        }
    return 1;
}
```

- Khi gọi *Search_Insert* trong chương trình chính mà h mang trị true thì điều đó có nghĩa là việc tách trang lan truyền đến trang gốc và trang gốc có thể bị tách trang.

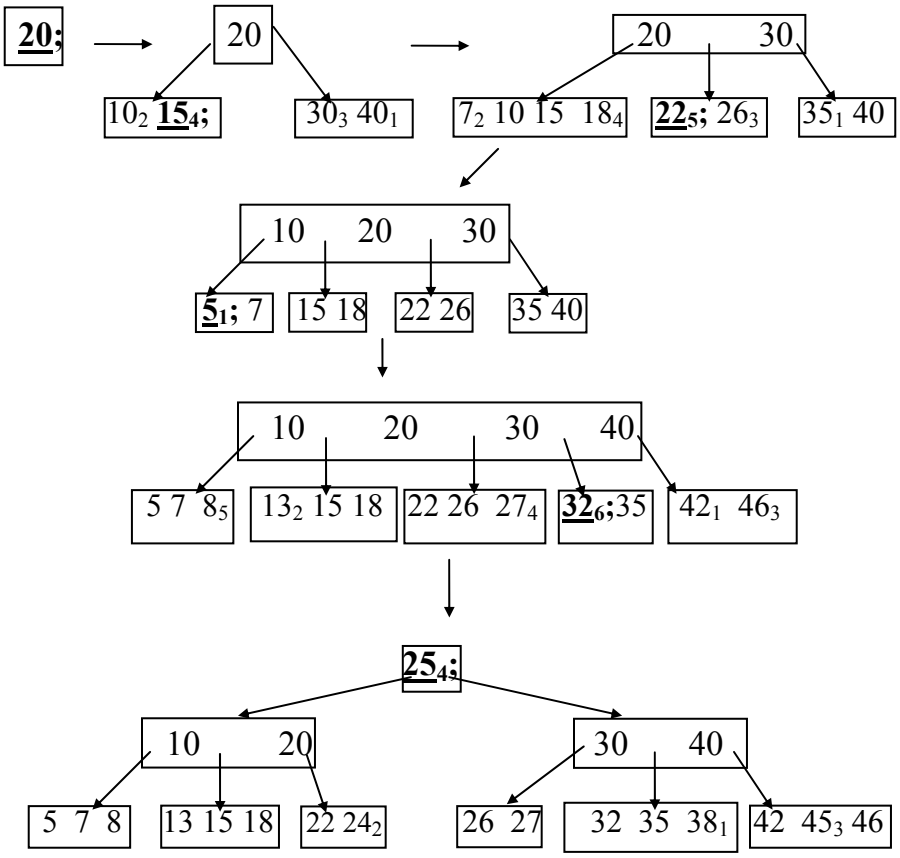
- Vì trang gốc có vai trò đặc biệt nên việc tách trang gốc được lập trình riêng, nó bao gồm việc cấp phát trang mới và thêm vào một phần tử, kết quả là trang gốc mới chỉ chứa 1 phần tử.

IV.2. <u>Giải thuật xây dựng B - cây</u>

Việc xây dựng B – cây cấp *n* bao gồm việc khởi tạo B – cây rỗng vào việc gọi liên tiếp thủ tục *Search_Insert* trên đây.

```
int XâyDựng_B_Cây(Ref   &root)
{int h;  Item  u; KeyType  x;
        root = NULL;
        while (CònNhậpLiệu(x))
        { if (!Search_Insert(x,root,h,u)) return 0;
          if (h) // chiều cao của cây tăng
                { Ref  q = root;
                  if (! CấpPhátTrang(root,n)) return 0;
                  root->m = 1;          // số phần tử của trang gốc mới bằng 1
                  root->p0 = q;
                  root->e[1] = u;       // u là phần tử được đưa lên trên trang cha root
                }
        }
        return 1;
}
```

***** *Ví dụ:* Tạo một B - cây cấp 2 từ dãy khoá sau: 20; 40, 10, 30, 15; 35, 7, 26, 18, 22; 5; 42, 13, 46, 27, 8, 32; 38, 24, 45, 25. Các dấu '**;**' chỉ ra các vị trí **"đột biến"** mỗi khi có sự cấp phát trang. Các bước tạo chính khi có sự tách trang là:



(Hình 1)

V. Xóa một phần tử khỏi B - cây

V.1. Hai tình huống loại bỏ một khóa trên B-cây

+ *Phần tử cần loại bỏ* ***thuộc trang lá***: việc loại bỏ diễn ra đơn giản.
+ *Phần tử cần loại bỏ* ***không thuộc trang lá***: việc loại bỏ phức tạp hơn:

- Trong tình huống thứ 2, *phần tử cần loại bỏ được* ***thay thế bởi 1 trong 2 phần tử kề nó*** *(về mặt giá trị) nằm ở trang lá* và có thể loại bỏ dễ dàng.
- Việc *tìm phần tử kề được thực hiện bằng cách đi dọc trên nhánh con trái theo các con trỏ cực phải đến trang lá P và phần tử kề là phần tử mút phải trên trang P*. Thay phần tử cần loại bỏ bởi phần tử này và giảm kích thước trang P đi 1.
- Sau khi giảm, kiểm tra số phần tử m trên trang P. *Nếu m < n thì cấu trúc B - cây bị vi phạm*. Khi đó, thực hiện các thao tác để xử lý tình trạng bị vi phạm này (trong trường hợp này dùng biến h kiểu boolean để chỉ ra điều kiện cạn này - *underflow* condition).

- Để xử lý ***trang bị cạn***, người ta "nối" một phần tử thuộc trang lân cận. Ta gọi Q là trang anh em bên trái hay bên phải của trang P. Ta *phân bố đều các phần tử trên cả 2 trang P và Q. Việc này gọi là "làm Cân Bằng"* (*balancing*).
- Tuy nhiên có thể có trường hợp *trang anh em Q chỉ còn n phần tử,* (*lúc này tổng số phần tử trên trang P và Q là 2n-1).* Khi đó ta *trộn (merge) 2 trang thành 1 trang P, cộng thêm 1 phần tử giữa lấy từ trang cha của trang P và Q, sau đó bỏ trang Q*. Đây là quá trình ngược của sự tách trang.
- Một lần nữa, việc lấy đi một phần tử thuộc trang cha có thể làm cho kích thước trang < n (bị cạn). Khi đó cần phải cân bằng hay trộn trang ở mức thấp hơn và quá trình này có khả năng lan truyền đến trang gốc. Nếu kích thước trang gốc giảm xuống 0 thì bỏ trang gốc và như vậy chiều cao của cây bị giảm đi. Đây là cách duy nhất làm giảm chiều cao của B-cây.

V.2. Giải thuật loại bỏ một khóa trên B-cây

. Input: - x: khoá cần tìm để xóa.
- a: trang hiện thời đang tìm.

. Output: Nếu h == True: cho biết gặp tình trạng bị cạn cần phải điều chỉnh với trang kế hoặc trộn lại

Delete (KeyType x, Ref &a, Boolean &h)

```
{       if (a == NULL)      // x không có trên cây
                h = 0;
        else    { // tìm kiếm x trên trang a
                        "Tìm kiếm nhị phân ";
                        "Cho q là trang con trái của e[k] trong trang a";
                        /*Trang q được xác định sau khi tìm nhị phân:
```

```
                    x = e[k].key hoặc e[k -1].key < x < e[k].key */
                if  "Tìm thấy "
                {       // tìm thấy x ở vị trí k: - xóa e[k] của trang a
                        if   ("q là trang lá")
                                "Bỏ e[k], giảm m đi 1, gán h bởi m < n";
                        else
                        {Del(a,q,k,h); // tìm phần tử  bị xóa thế x
                         if (h) "Điều chỉnh với trang kế hoặc trộn lại";
                                                // Underflow
                        }
                }
                else
                { // không tìm thấy
                        Delete(x,q,h);
                        if (h) "Điều chỉnh với trang kế hoặc trộn lại";
                                                // Underflow
                }
        }
    return ;
}
```

- Thủ tục Underflow thực hiện việc "Điều chỉnh với trang kế hoặc trộn lại".
- Thủ tục Del thực hiện việc thay thế phần tử mép phải (cuối cùng) của trang lá cực phải cho phần tử cần bị xóa x=e[k].key.

Del (Ref a, Ref p, int k, Boolean &h)

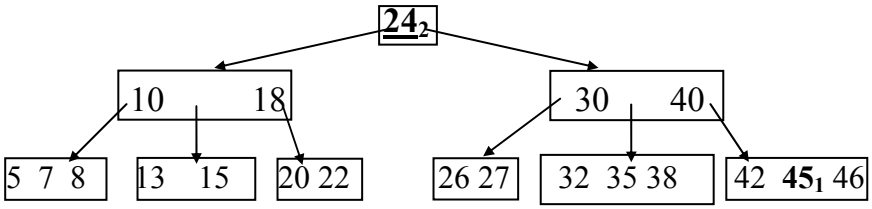
```
{       q = trang con phải của phần tử mép phải của p;
        if  "q không phải là trang lá"
        {       Del (a,Trang_Con_Cuối(p), k, h);
                if (h)  "Điều chỉnh với trang kế hoặc trộn lại"; // Underflow
        }
        else
        { "Thay phần tử cuối cùng của trang p vào phần tử bị loại bỏ e[k],
           giảm m đi 1, gán h bởi m < n";
        }
        return ;
}
```

***** *Ví dụ:* Xét B-cây cấp 2 của hình 1 trong phần IV.2. Lần lượt loại bỏ các khóa: 25, 45, **24**; 38, **32**; 8, 27, 46, 13, **42**; 5, 22, 18, **26**; 7, 35, **15**;

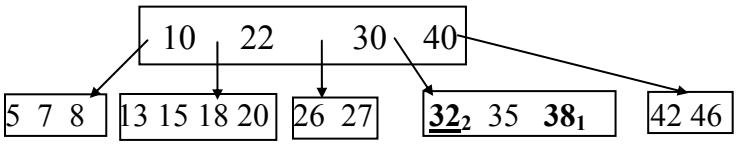
Dấu '**;**' chỉ ra các vị trí "**đột biến**" tại đó có trang bị loại bỏ. (*Giả sử*, ta chọn *phần tử thay thế* ***kề trái*** *của x* được chọn là phần tử *lớn nhất mà vẫn <x*. Việc *điều chỉnh với trang kế* xảy ra với *trang anh em (ruột: chúng cùng trang cha) kề*

phải trước, *nếu không thể mới xét anh em kề trái*; sau đó, nếu không thể, mới *sát nhập với anh em kề phải trước*; nếu không có anh em phải mới đến lượt sát nhập anh em kề trái).

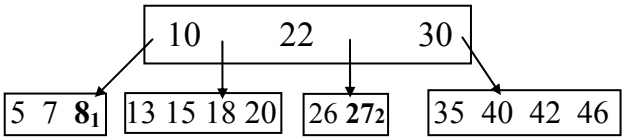
+ Cây sau khi loại bỏ khoá 25:



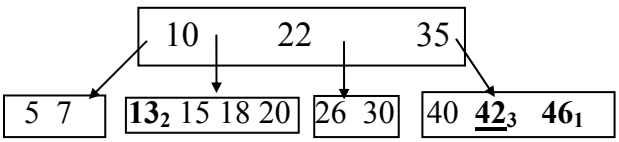
+ Cây sau khi loại bỏ các khoá 45, **24**:



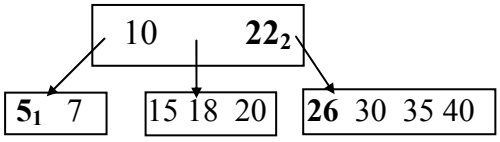
+ Cây sau khi loại bỏ các khoá 38, **32**:



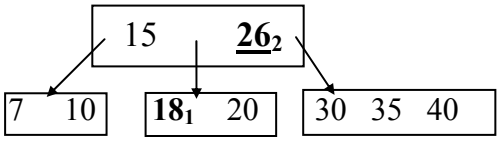
+ Cây sau khi loại bỏ các khoá 8, 27:



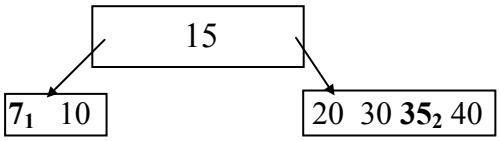
+ Cây sau khi loại bỏ các khóa 46, 13, **42**:



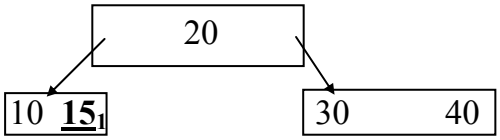
+ Cây sau khi loại bỏ các khóa 5, 22: (tìm 20 kề trái trước: bị cạn ! → mượn phần tử trái nhất 26 trong trang anh em ruột kề phải)



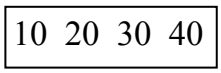
+ Cây sau khi loại bỏ các khóa 18, **<u>26</u>**:



+ Cây sau khi loại bỏ các khóa 7, 35:



+ Cây sau khi loại bỏ khóa **<u>15</u>**:



Chương 3

BẢNG BĂM

I. Đặt vấn đề, mục đích, ý nghĩa

Một trong những nhiệm vụ chính của tin học, ngoài việc *lưu trữ* thông tin, là *khai thác* và *xử lý* thông tin. Trong việc khai thác thông tin, việc tìm kiếm đóng vai trò quan trọng. Ngoài các phương pháp tìm kiếm đã biết như tìm kiếm tuyến tính trên dãy các đối tượng chưa sắp hay tìm kiếm nhị phân trên dãy các đối tượng đã sắp, người ta còn xét các phương pháp khác rất hiệu quả. *Phương pháp biến đổi khoá là một phương pháp tìm kiếm hữu hiệu* như vậy.

Sở dĩ các phương pháp tìm kiếm thông thường theo giá trị khóa không thật hiệu quả là do, trong các phương pháp này, việc *truy nhập đến một đối tượng trong mảng ít liên quan trực tiếp đến chính bản thân giá trị khóa của đối tượng* đó.

Phương pháp biến đổi khóa (key transformation) là phương pháp *tham khảo trực tiếp các đối tượng trong bảng thông qua phép biến đổi số học trên những khoá (key) để biết được địa chỉ tương ứng của các đối tượng* trong bảng. Khi áp dụng các phương pháp biến đổi khóa trong việc xây dựng dãy các đối tượng trong bảng và tìm kiếm một đối tượng trên bảng đó, ta phải *tốn thêm thời gian cho các phép biến đổi số học* trên những khóa và cho việc *giải quyết tình trạng đụng độ* (tình trạng *nhiều khóa khác nhau* nhưng lại *có cùng giá trị qua phép biến đổi khóa*).

II. Phương pháp biến đổi khóa

Xét dãy các đối tượng có cùng kiểu *T*, để truy nhập đến một đối tượng thuộc dãy ta cần biết địa chỉ của nó. Gọi ***A*** là miền trị của các địa chỉ này. Giả sử trong kiểu *T*, có một trường khóa (key), sau khi số hóa nếu cần thiết, thuộc vào miền trị ***K*** nào đó.

Phép biến đổi khoá là một ánh xạ thích hợp H từ tập các khóa K đến tập các địa chỉ A:

$$\mathbf{H: K \rightarrow A}$$

Không giảm tính tổng quát và để đơn giản trong trình bày, ta *giả sử dãy các đối tượng được lưu trong một mảng (băm)*. Khi đó H là *ánh xạ biến đổi khóa thành chỉ số trong mảng*.

Trong thực tế ta hay gặp trường hợp tập các giá trị khóa có số lượng lớn hơn rất nhiều so với tập các địa chỉ bộ nhớ (chẳng hạn tập chỉ số của mảng). Khi đó, *H là ánh xạ nhiều-một (H không đơn ánh)*.

* *Ví dụ 1*: Ta dùng một tập các khóa mà mỗi khóa gồm 10 ký tự để định danh cho tập gồm 1000 người. Với bộ ký tự có 26 ký tự chữ cái, khi đó tập khóa có 26^{10} trị khác nhau được ánh xạ vào tập gồm 10^3 chỉ số. Lúc đó có thể xảy ra *tình trạng đụng độ*: 2 khóa khác nhau có thể cho cùng một chỉ số qua một phép biến đổi khóa H nào đó.

Phương pháp biến đổi khóa gồm hai giai đoạn:

- *Giai đoạn 1*: *Chọn phép biến đổi khóa H* và *tính trị hàm H* tại trị khóa của một đối tượng để xác định địa chỉ của đối tượng trong mảng.
- *Giai đoạn 2*: *Giải quyết tình trạng đụng độ* (collision resolution) cho những *khóa khác nhau có cùng một địa chỉ* trong mảng. Ta thường *giải quyết đụng độ* bằng cách *dùng các danh sách liên kết (*hoặc *tốt hơn* khi dùng *cây tìm kiếm nhị phân BST, cây cân bằng AVL, B cây, ...)* để *lưu các đối tượng có cùng địa chỉ băm* trong mảng, do ta không biết trước các số lượng những đối tượng có tính chất này. Một *phương pháp khác* để giải quyết đụng độ với thời gian *nhanh* là *dùng mảng có kích thước lớn và cố định* như trong phương pháp *địa chỉ mở*.

III. Hàm biến đổi khóa (hàm băm)

Yêu cầu của phép biến đổi khóa là khả năng *phân bố đều trên miền trị của địa chỉ*. Do yêu cầu này mà phương pháp (hàm) biến đổi khóa còn được gọi là phương pháp (hàm) băm (hash).

Gọi *M* là số các phần tử của mảng chứa các địa chỉ (hay chỉ số, vị trí). Hàm băm thường biến đổi các khóa (thường là các số tự nhiên hoặc các chuỗi ký tự ngắn) thành các số nguyên không âm trong đoạn *[0.. M-1]*. *Với đối tượng có khóa k, giá trị H(k) (0 ≤ H(k) ≤ M-1) được dùng làm cơ sở để lưu trữ cũng như tìm kiếm đối tượng.*

Giả sử các khóa *k* là các số nguyên không âm, ta thường dùng hàm băm:

H[k] = k mod M

Do tính chất số học của hàm mod, ta thường *chọn M là số nguyên tố* để giảm bớt tình trạng đụng độ.

* *Ví dụ 2*: Để *số hóa giá trị khóa là chuỗi các ký tự* chữ cái alphabet, ta dùng 5 bits để mã hóa mỗi ký tự (ký tự thứ i trong bảng thứ tự alphabet được mã

thành số nhị phân tương ứng với số i). Mỗi chuỗi ký tự được mã hóa bằng cách đặt các dãy 5 bits này liên tiếp nhau, ta thu được một số (theo biểu diễn cơ số $2^5 = 32$). Chẳng hạn, với chuỗi: AKEY

Ký tự	Thập phân	Nhị phân
A	1	00001
B	2	00010
…	…	…
E	5	00101
…	…	…
K	11	01011
…	…	…
Y	25	11001
…	…	…

Ta biểu diễn chuỗi AKEY bằng dãy bits:

00001 01011 00101 11001

hay tương đương với số sau theo cách biểu diễn trong hệ cơ số 32:

$k_0 = 1*32^3 + 11*32^2 + 5*32^1 + 25*32^0$

Nếu *chọn M = 32 (không nguyên tố*) thì hàm băm H(k) = k mod M chỉ phụ thuộc vào ký tự cuối cùng:

$H(k_0) = 25 \bmod 32 = 25$

* *Chú ý*: Nếu *khóa k[keysize] là chuỗi các ký tự (chữ hay số) dài*, để *tránh* tình trạng *tính toán lâu* và thậm chí *bị tràn số*, ta có thể dùng *thuật toán Horner* để tính trị hàm băm cho khóa *k* sau khi mã hoá (số hóa k theo cơ số b, chẳng hạn bằng 32, với *k[i]* được hiểu là số thứ tự của ký tự đó trong bảng chữ cái):

$$\text{SốHóa}(k) = \sum_{i=0}^{keysize-1} k[i] * b^{keysize-i-1}$$

$$= (\ldots(((k[0]*b) + k[1])*b + \ldots k[keysize-2])*b + k[keysize-1]$$

$$H(k) = H1(\text{SốHóa}(k)) = \text{SốHóa}(k) \bmod b$$

```
nguyên H1(nguyên k[keysize], int co_so, int M)
{ nguyên h=k[0];
      for (int i=1; i< keysize; i++)  h = (h * co_so  +  k[i])  mod M;
      return  h;
}
```

Hãy tìm cơ sở để chứng minh tính đúng của thuật toán (*bài tập*).

IV. Giải quyết sự đụng độ

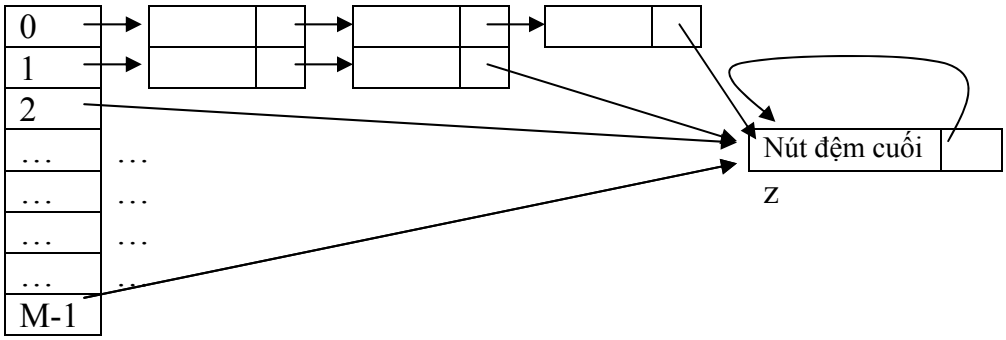
Khi dùng hàm băm có thể sẽ dẫn đến tình trạng *đụng độ*: có (ít nhất) 2 khoá khác nhau k1 ≠ k2 nhận cùng địa chỉ băm (trị của hàm băm): H(k1) = H(k2). Để *khắc phục tình trạng đụng độ*, ta có thể dùng phương pháp *băm liên kết* (thông qua danh sách liên kết được minh họa bằng con trỏ hoặc mảng liên kết) hoặc băm theo phương pháp *địa chỉ mở.*

IV.1. Phương pháp băm liên kết

Trong phương pháp *băm liên kết,* ta dùng một mảng *những danh sách liên kết, mỗi danh sách chứa các đối tượng có cùng địa chỉ băm.*

IV.1.1. Phương pháp băm liên kết trực tiếp

Trong phương pháp *băm liên kết* ***trực tiếp****,* ta dùng loại *danh sách liên kết được cài đặt bằng con trỏ, danh sách này được kết thúc bằng NULL* hoặc *có nút đệm ở cuối* (đóng vai trò phần tử *lính canh trong các thao tác tìm kiếm* sau này). Khi *xây dựng bảng các địa chỉ băm*, nếu phải đưa thêm một đối tượng mới vào một danh sách liên kết ứng với cùng một địa chỉ băm nào đó, nên chèn nó vào vị trí thích hợp để *danh sách liên kết này được sắp thứ tự*, phục vụ cho việc *tìm kiếm* sau này *nhanh hơn*. *Kích thước bảng băm M có thể bé hơn kích thước n của dãy chứa các đối tượng cần lưu*.

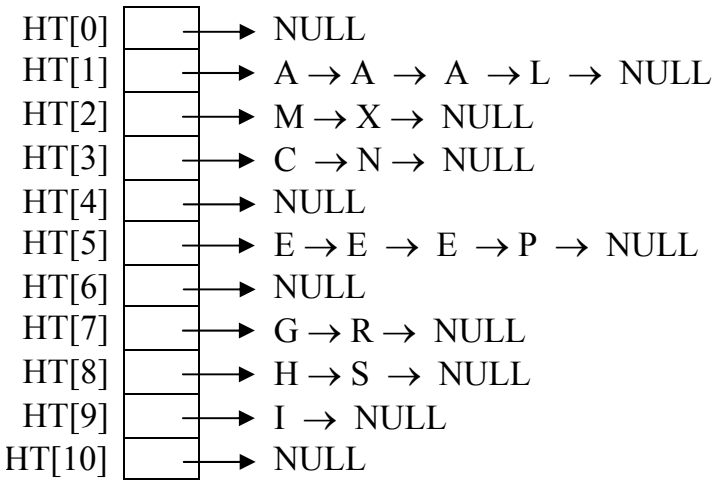


* *Ví dụ 3*: Xét dãy các khóa là các ký tự k (SốHóa(k) = ChữHoa(k)-'A'+1: là số thứ tự của chữ cái k trong bảng chữ cái alphabet) và trị hàm băm tương ứng H(k) = SốHóa(k) mod M được lần lượt đưa vào bảng băm với kích thước M = 11 như sau:

. Khóa k: A S E A R C H I N G E X A M P L E

. Hàm băm H(k): 1 8 5 1 7 3 8 9 3 7 5 2 1 2 5 1 5

Sau khi chèn, ta sẽ có M danh sách liên kết như sau:



A. Thuật toán *xây dựng bảng băm liên kết trực tiếp* HT_LKTT cấp M, từ dãy n khóa được lưu trong mảng a:

- Input: M, n, a.

- Output: bảng băm HT_LKTT.

XâyDựngBảngBămLiênKếtTrựcTiếp(HT_LKTT, M, a, n)
{ for (i=0; i<M; i=i+1) HT_LKTT[i] = KhởiTạoDSLKRỗng();
for (i=0; i<n; i=i+1) *ChènTăngVàoDSLK*(HT_LKTT[H(a[i])], a[i]);
}

trong đó: *ChènTăngVàoDSLK*(LL, x) là thủ tục chèn khóa x vào DSLK LL đã tăng thành LL mới vẫn tăng.

B. Thuật toán *tìm khóa k trong bảng băm liên kết trực tiếp* HT_LKTT cấp M:

- Input: bảng băm HT_LKTT, cấp M, khóa k.

- Output: true nếu k có trong HT_LKTT và false nếu ngược lại.

Bool **TìmKiếmĐốiTượngTrongBảngBămLiênKếtTrựcTiếp**(HT_LKTT, M, k)
{ return *TìmKiếmTrongDSLKTăng*(k, HT_LKTT[H(k)]);
}

trong đó: *TìmKiếmTrongDSLKTăng*(k, LL) là thủ tục tìm khóa k trong DSLK LL tăng.

C. Thuật toán *xóa khóa k trong bảng băm liên kết trực tiếp* HT_LKTT cấp M:

- Input: bảng băm HT_LKTT, cấp M, khóa k.

- Output: true nếu xóa được k (có) trong HT_LKTT và false nếu ngược lại.

Bool **XóaĐốiTượngTrongBảngBămLiênKếtTrựcTiếp**(HT_LKTT, M, k)
{ return *XóaTrongDSLKTăng*(k, HT_LKTT[H(k)]);
}

trong đó: *XóaTrongDSLKTăng*(k, LL) là thủ tục xóa khóa k trong DSLK LL tăng. Các thủ tục *chèn, xóa, tìm kiếm* các đối tượng trên danh sách liên kết đã được trình bày trong giáo trình "*Cấu trúc dữ liệu và giải thuật 1*".

IV.1.2. Phương pháp băm liên kết kết hợp

Trong phương pháp băm liên kết này, *danh sách liên kết* ***được cài đặt bằng một mảng*** các cấu trúc, với mỗi cấu trúc *ngoài thành phần dữ liệu* ***Data*** thông thường của đối tượng, còn *có thêm một trường* ***Next*** *chứa địa chỉ (chỉ số trên mảng) của đối tượng kế tiếp khi có sự đụng độ xảy ra*. Điều kiện cần thỏa mãn để thực hiện phương pháp này là: *kích thước mảng M phải lớn hơn n - số thành phần dữ liệu của dãy cần lưu* !

* *Ví dụ 4*: Giả sử các khoá có trị hàm băm và thứ tự thêm vào như sau:

Khóa: A C B D

Hash: 0 1 0 0

Gọi hằng: *Rỗng =-2, KếtThúc = -1 (*là chỉ số kết thúc của dãy các khóa có cùng giá trị băm). Đầu tiên, ta khởi tạo bảng băm HT chứa toàn các vị trí trống HT[i].Next = Rỗng, i = 0..M-1, khởi tạo chỉ số trống đầu tiên từ dưới lên: r =M-1:

	Data	Next
HT[0]	?	Rỗng
HT[1]	?	Rỗng
…	…	…
HT[M-2]	?	Rỗng
HT[M-1]	?	Rỗng

Sau khi thêm lần lượt 4 khóa trên, ta có bảng băm *HT* như sau:

	Data	Next
HT[0]	A	M-1
HT[1]	C	KếtThúc
..	…	…
HT[M-2]	D	KếtThúc
HT[M-1]	B	M-2

Đầu tiên, do H(A)=0 (HT[0].next = Rỗng), nên ta đặt A vào HT[0]: HT[0].key = A, HT[0].next = KếtThúc.

Tương tự, HT[1].key = C, HT[1].next = KếtThúc.

Đáng lẽ ta đặt B vào HT[0], nhưng tại đó đã có A (HT[0].next ≠ Rỗng, gặp đụng độ !), nên ta phải tìm vị trí trống (từ đưới lên) r = M-1 để đặt B vào đó: HT[r].key = B, HT[r].next = KếtThúc và sửa lại chỉ mục của A: HT[0].next = r = M-1. Lúc đó, vị trí trống đầu tiên từ dưới lên là: r ← r-1 = M-2.

Lại đưa thêm D, do: H(D)=0, HT[0].next = M-1 ≠ Rỗng, lại xét tiếp (cho đến khi): HT[M-1].next = KếtThúc. Khi đó: ta sẽ đặt D vào vị trí trống đầu tiên từ dưới lên r = M-2: HT[r].key = D, HT[r].next = KếtThúc và sửa lại chỉ mục (KếtThúc) tại HT[M-1]: HT[M-1].next = r = M-2. Lúc đó, vị trí trống đầu tiên từ dưới lên là: r ← r-1 = M-3.

* *Nhận xét 1*: Khi thêm vào bảng băm, các phần tử HT[j] đều đã sử dụng, với mọi j = r+1.. M-1.

A. ***Cấu trúc dữ liệu*** *cho bảng băm liên kết kết hợp* HT_LKKH:

```
struct Node { DataType Data;
                Nguyen Next;
              };
struct { Node HTLKKH[MaxPhanTu]; // mảng chứa dữ liệu và liên kết
         Nguyen M; // cấp hay kích thước bảng băm
         Nguyen SoPT; // số khóa dữ liệu đã chèn vào bảng băm
         Nguyen ViTriTrong; // vị trí trống từ dưới lên r
       } HT;
```

B. Thuật toán *xây dựng bảng băm liên kết kết hợp* HT cấp M, từ dãy n khóa được lưu trong mảng a:

- Input: a, n.

- Output: bảng băm HT.

XâyDựngBảngBămLiênKếtKếtHợp(HT, a, n)

```
{       for (i=0; i<HT.M; i=i+1) HT.HTLKKH[i].Next = Rỗng;
        HT.ViTriTrong = HT.M - 1;
        HT.SoPT = 0;
        for (i=0; i<n; i=i+1) if (!ChenPTVao_HTKH(a[i], HT)) return;
}
```

trong đó `ChenPTVao_HTKH`(k, HT) là thủ tục chèn khóa k vào HT:

```
Bool ChenPTVao_HTKH(k, HT)
{
      if (HT.SoPT==HT.M-1)
         { Xuất("\nBang bam day"); return false;
         }
      VT=k mod HT.M;
      if (HT.HTLKKH[VT].Next==Rỗng)
      {     HT.HTLKKH[VT].Data = k; HT.HTLKKH[VT].Next = KếtThúc;
            if (VT==HT.ViTriTrong) Tim_ViTriTrongKeTiep(HT);
      }
      else
      { while (HT.HTLKKH[VT].Next != KếtThúc)
        VT = HT.HTLKKH[VT].Next;
```

```
        HT.HTLKKH[HT.ViTriTrong].Data = k;
        HT.HTLKKH[HT.ViTriTrong].Next = KếtThúc;
        HT.HTLKKH[VT].Next = HT.ViTriTrong;
        Tim_ViTriTrongKeTiep(HT);
      }
      HT.SoPT = HT.SoPT +1;
      return true;
}
```

// Hàm tìm vị trí trống kế tiếp từ dưới lên:

```
Tim_ViTriTrongKeTiep(HT)
{     do
      { HT.ViTriTrong = HT.ViTriTrong-1;
      }
      while (HT.ViTriTrong>=0 && HT.HTLKKH[HT.ViTriTrong].Next!=Rong);
}
```

C. Thuật toán *tìm khóa k trong bảng băm liên kết kết hợp* HT cấp M:

- Input: bảng băm HT, khóa k.

- Output: true nếu k có trong HT và false nếu ngược lại.

Bool **TìmKiếmĐốiTượngTrongBảngBămLiênKếtKếtHợp**(HT, k, VTriThay)

```
{   Ke = k mod HT.M;
    if (HT.HTLKKH[Ke].Next==Rong) return false;
    else
    do {  if (HT.HTLKKH[Ke].key==k)
          { VTriThay = Ke;
            return true;
          }
          if (HT.HTLKKH[Ke].Next!=KếtThúc)
               Ke=HT.HTLKKH[Ke].Next;
          else break;
       }while (1);
    return false;
}
```

**** Nhận xét***:

- Có thể áp dụng các phương pháp băm liên kết này để tổ chức *lưu trữ các đối tượng trên file*, phục vụ cho việc *tìm kiếm ngoài (bài tập)*.

- Nếu *số phần tử trong mỗi danh sách (chứa các khóa có cùng trị băm) khá lớn*, ta có thể *thay* các *danh sách liên kết* bởi các *cây nhị phân tìm kiếm BST, cây cân bằng AVL* hoặc *B cây* thì sẽ thu được *phương pháp băm hiệu quả hơn nhiều (bài tập).*

- Để đơn giản việc *xóa khóa k* khỏi bảng băm liên kết trực tiếp, ta có thể thêm một *trường đánh dấu xóa* (kiểu lô-gic) vào cấu trúc Node của một nút. Khi đó, sau khi xóa, ta còn có thể phục hồi lại chúng (*bài tập*).

IV.2. Băm theo phương pháp địa chỉ mở

Phương pháp *liên kết trực tiếp có nhược điểm* là phải *sử dụng thêm trường liên kết next* trong mỗi nút của danh sách liên kết.

Một cách giải quyết đụng độ khác là phương pháp *địa chỉ mở* hay *băm thử.*

A. *Cấu trúc dữ liệu* *cho bảng băm thử* Open_HT:

```
struct { DataType OpenHT[Max];
         Nguyen M; // cấp bảng băm
         Nguyen SoPT; // số khóa dữ liệu đã chèn vào bảng băm
       } HT ;
```

B. *Xây dựng bảng băm thử*: Khi ***lưu trữ*** một khóa, nếu có đụng độ xảy ra thì ta sẽ *tìm đến vị trí kế tiếp* nào đó trong bảng *dựa theo dãy chỉ số ở bước thử thứ hai* (để xác định vị trí kế tiếp) *cho đến khi tìm thấy phần tử ở vị trí kế tiếp này là trống* (trùng với một hằng *free* đặc biệt nào đó, nằm ngoài miền trị *K* của khóa, dùng để nhận biết các vị trí trống trong bảng băm). *Dãy chỉ số ở bước thử thứ hai* phải *luôn giống nhau* đối *với một khóa cho trước*.

Gọi *M (nguyên tố*) là số phần tử của bảng băm, *n* là số phần tử *đã sử dụng (**n < M**)*. Bảng băm gọi là ***đầy*** khi N=M-1. Như vậy, bảng *băm luôn có ít nhất một phần tử trống (nếu cần thiết, dùng làm lính canh* trong các thuật toán tìm, chèn, xóa, … sau này*)*. Gọi:

. H(k) là trị hàm băm tại khóa *k* (để *biết vị trí của khóa k trong bảng băm*),

. G(j, k) là hàm tạo ra *dãy chỉ số của phép thử thứ hai*.

Thuật toán *xây dựng bảng băm địa chỉ mở* HT cấp M, từ dãy n khóa được lưu trong mảng a:

- Input: a, n.

- Output: bảng băm địa chỉ mở HT.

```
XayDung_BangBamThu(HT, a, n)
{//Khởi tạo bảng băm thử rỗng:
     for (i=0; i<HT.M; i=i+1) HT.OpenHT[i]= free;
     HT.SoPT = 0;
 //Chèn các phần tử trên dãy a vào bảng băm:
     for (i = 0; i<n; i=i+1)
          if (!Chen_1MucDuLieu_OpenHT(HT, a[i])) return;
}

bool Chen_1MucDuLieu_OpenHT(HT, k)
```

```
{    ViTriDau=H(k,HT.M); j=0; ViTriKe=ViTriDau;
     if (HT.SoPT == HT.M)
     {    Xuất("Bang Bam Day!"); return false;
     }
     ViTriKe =(ViTriDau + G(j,k)) mod HT.M;
     while (HT.OpenHT[ViTriKe] != free)
     {    j = j+1;
          ViTriKe=(ViTriDau + G(j,k)) mod HT.M;
     }
     HT.OpenHT[ViTriKe] = x; HT.SoPT = HT.SoPT+1;
     return true;
}
```

C. *Tìm kiếm trên bảng băm thử*:

Khi ***tìm kiếm*** một khóa *k*, nếu phần tử tại vị trí *H(k)* là:

. phần tử cần tìm thì giải thuật tìm kiếm kết thúc thành công (tìm thấy);

. *free* thì giải thuật tìm kiếm kết thúc không thành công (không tìm thấy);

. không phải là *free* và khác phần tử cần tìm thì dựa vào hàm băm thứ hai, ta tiếp tục tìm ở vị trí kế tiếp

Giải thuật ***tìm*** *khóa k theo phương pháp địa chỉ mở* như sau:

- Input: bảng băm thử HT, khóa k.

- Output: true (và VịTrí trên HT thấy k) nếu thấy k trong HT; false nếu ngược lại.

Bool ***TìmTheoĐịaChỉMở***(*k*, HT, &ViTri)

{ BatDau ← *H[k];* ViTri ← BatDau; j ← 0;
while (HT.OpenHT[ViTri] ≠ *free* and HT.OpenHT[ViTri] ≠ *k*)
{ ***j ←j+1;*** *ViTri ← (BatDau+G(j, k)) mod HT.*M*;*
}
if (HT.OpenHT[ViTri] == k) return true;
else return false;
}

* ***Nhận xét 2***: a. Hàm *G lý tưởng* là hàm có thể *trải đều các khóa trên các vị trí còn lại* nhưng lại *không cần phải tính toán quá lâu* !

b. Hai hàm H và G cần thỏa *điều kiện* sau để *khi gặp đụng độ*, *luôn tìm được vị trí chèn vào bảng băm nếu nó còn trống*:

∀k,∀i=0..M-1, ∃j≥0: (H(k)+G(j, k)) mod M = i **(1)**

⇔ ∀k, H(k)+G(ℕ, k)⊇[0..M-1]. **(1')**

c. *Điều kiện đủ cho:*

$\forall i,x = 0..M-1, \exists j \geq 0: (x+j*z) \bmod M = i$ **(2)**

là: **(M,z) = 1** (M và z nguyên tố cùng nhau: *thường chọn 0<z<M*) **(2')**

hoặc: **M nguyên tố** và ***0<z<M*** **(2'')**

d. *Điều kiện đủ cho (1)* là:

$\mathbf{G(j, k) = j*H_2(k), \forall k: H_2(k) \neq mM}$ và **M nguyên tố** **(3)**

e. Với ***các hàm G(j,k) khác nhau***, *ta sẽ nhận được* ***các phương pháp băm khác nhau*:**

- Phương pháp **băm thử tuyến tính: G(j,k) ≡ j** *(không phụ thuộc vào khóa k)*
- Phương pháp **băm kép: G(j,k) ≡ j*H₂(k)** *(phụ thuộc hàm băm thứ hai* $\mathbf{H_2}$*(k))*
- Phương pháp **băm thử bậc hai: G(j,k) ≡ j²** *(không phụ thuộc vào khóa k)*

Chứng minh (bài tập).

IV.2.1. <u>Phương pháp băm (thử) tuyến tính</u>

Phương pháp băm theo **địa chỉ mở đơn giản** là dùng ***phép thử tuyến tính***, với **G(j, k) ≡ j** , có nghĩa là khi đụng độ xảy ra, ta tìm đến vị trí kế tiếp (chỉ số tìm kiếm trên mảng tăng lên 1, *khi đó điều kiện (1) luôn thỏa*).

Dãy chỉ số x_j dùng để thử là:

$x_0 = \mathbf{H(k)}$

$x_j = (x_0 + j) \bmod M$, *với mọi j=1.. M-1*

* <u>*Ví dụ 5*</u>: Xét dãy các khóa và trị hàm băm tương ứng được lần lượt đưa vào bảng băm với kích thước M = 19 như sau:

Khóa: A S E A R C H I N G E X A M P L E

Hash: 1 0 5 1 18 3 8 9 14 7 5 5 1 13 16 12 5

Sau khi chèn, ta có bảng *H.1* dưới dây (trong đó ~ ký hiệu vị trí *rỗng*).

* <u>*Nhận xét*</u>:

- *Kích thước bảng băm* của phương pháp địa chỉ mở *phải lớn hơn kích thước bảng băm* trong phương pháp liên kết trực tiếp (vì M>n); nhưng *vùng nhớ tổng cộng* của phương pháp địa chỉ mở *lại có thể nhỏ hơn* vì nó không tốn vùng liên kết.

- Gọi a = n/M là ***hệ số tải*** của bảng băm. ***Số lần so sánh trung bình*** trong trường hợp:

  . *tìm kiếm không thành công* là:

  $$C1 = \tfrac{1}{2} + 1/(2 * (1 - a)^2)$$

  . *tìm kiếm thành công* là:

  $$C2 = \tfrac{1}{2} + 1/(2 * (1 - a))$$

- Nếu *a = 2/3* thì trung bình ta cần:

  . *5 lần* so sánh trong trường hợp tìm kiếm *không thành công*

. *2 lần* so sánh trong trường hợp tìm kiếm *thành công*

- Nếu *a gần 1* (*bảng băm gần đầy*) thì *số lần so sánh* trung bình sẽ *tăng rất nhanh*.

HT	***A***	***S***	***E***	***A***	***R***	***C***	***H***	***I***	***N***	***G***	***E***	***X***	***A***	***M***	***P***	***L***	***E***
0	~	**S**	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
1	**A**	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
2	~	~	~	**A**	A	A	A	A	A	A	A	A	A	A	A	A	A
3	~	~	~	~	~	**C**	C	C	C	C	C	C	C	C	C	C	C
4	~	~	~	~	~	~	~	~	~	~	~	~	**A**	A	A	A	A
5	~	~	**E**	E	E	E	E	E	E	E	E	E	E	E	E	E	E
6	~	~	~	~	~	~	~	~	~	~	**E**	E	E	E	E	E	E
7	~	~	~	~	~	~	~	~	~	**G**	G	G	G	G	G	G	G
8	~	~	~	~	~	~	**H**	H	H	H	H	H	H	H	H	H	H
9	~	~	~	~	~	~	~	**I**	I	I	I	I	I	I	I	I	I
10	~	~	~	~	~	~	~	~	~	~	~	**X**	X	X	X	X	X
11	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	**E**
12	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	**L**	L
13	~	~	~	~	~	~	~	~	~	~	~	~	~	**M**	M	M	M
14	~	~	~	~	~	~	~	~	**N**	N	N	N	N	N	N	N	N
15	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~
16	~	~	~	~	~	~	~	~	~	~	~	~	~	~	**P**	P	P
17	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~
18	~	~	~	~	**R**	R	R	R	R	R	R	R	R	R	R	R	R

(H.1)

IV.2.2. <u>Phương pháp băm (thử) bậc hai</u>

Trong phương pháp thử tuyến tính, khi bảng a gần đầy, thời gian tìm một vị trống kế tiếp khi có đụng độ có thể sẽ rất lâu. Chẳng hạn, trong ví dụ 5, khi thêm X (có trị băm là 5) thì gặp đụng độ. Ta cần đến 6 lần so sánh để đưa X vào vị trí 10. Trong những lần so sánh đó, ta phải so sánh X với những khóa không có cùng trị băm với nó như: E, G, H, I.

Trong trường hợp xấu nhất, *khi thêm* một phần tử có trị băm nào đó *có thể làm tăng đáng kể số lần tìm kiếm đối với những khóa có trị băm khác. Ta gọi đó là hiện tượng "**gom tụ**" (clustering), nó làm cho phương pháp thử tuyến tính được thực hiện rất chậm trong trường hợp bảng gần đầy !* Để tránh hiện tượng gom tụ này, ta có dùng phương pháp *thử bậc hai, bằng cách chọn:*

$$G(j, k) = j^2$$

Dãy chỉ số x_j dùng để thử là:

$x_0 = H(k)$

$x_j = (x_0 + j^2) \bmod M$, $\forall j=1..\ M-1$

Khi đó, *điều kiện (1) có thể không thỏa mãn!*

Để tính toán nhanh hơn, ta đặt:

$a_j = j^2$

$d_j = 2*j + 1.$

=> $a_{j+1} = a_j + d_j$

$d_{j+1} = d_j + 2$, *với* $a_0 = 0$ *và* $d_0 = 1$.

IV.2.3. Phương pháp băm kép

Một phương pháp khác để tránh hiện tượng gom tụ trong phương pháp băm tuyến tính là dùng phương pháp băm kép: *thay vì xét các vị trí kế tiếp (phụ thuộc vào khóa k) ngay sau vị trí đụng độ*, ta ***dùng hàm băm thứ hai $H_2(k)$*** *như một bước nhảy (khác nhau ứng với mỗi khóa k: G(j, k)=j*$H_2(k)$, j = 1,2, ...) được dùng trong các lần thử sau đó (*khi đó *thời gian tìm sẽ tăng lên* đặc biệt đối với *khóa dài)*.

Sau đây là thuật toán tìm kiếm và chèn một khóa *k* vào bảng băm *HT* có kích thước *M*. Hàm trả về vị trí tìm thấy hoặc vị trí thêm vào nếu giá trị hàm bé hơn M và trả về trị M nếu bảng băm đầy.

```
Nguyen Tìm_Chèn(khóa k, BảngBăm HT, KíchThước M)
    {   x ← H(k);
        y ← H2(k);
        while (HT[x].key ≠ free and HT[x].key ≠ k)
            x ← (x+y) mod M;
        if (HT[x].key == k)  return  x;          // tìm thấy
        else  if (n == M-1) return M;            // bảng băm đầy
              else                               // thêm khóa k vào bảng băm
                  { HT[x].key ← k;
                    n ← n+1;
                    return  x;
                  }
    }
```

* ***Một số điều cần lưu ý khi chọn hàm băm thứ hai H_2***:

- *Một cách lý tưởng, để thỏa điều kiện (1) và (3) trong nhận xét 2, nên chọn:*

M và $y = H_2(k)$ nguyên tố cùng nhau ***(4)***

(nên chọn 0 < y = $H_2(k)$ <M; vì nếu y = 0: chương trình bị rơi vào vòng lặp quẩn khi có khóa k_2 đụng độ với khóa k_1: y = $H_1(k)$ = $H_2(k)$ = 0; nếu y>0 không nguyên tố cùng nhau với M = 2*y: sẽ gặp những chuỗi phép thử rất ngắn; nếu y≠kM và y>M thì thay y bởi y* = y mod M, khi đó 0<y*<M)

- *Điều kiện (1) được thỏa nếu:*

  M nguyên tố và 0<y=$H_2(k)$<M **(5)**

- Nếu chọn hàm băm thứ hai:

  . *$H_2(k) = H(k)$,* thì *$G(j, k) = j*H(k)$,* j = 1,2, …

  . *$H_2(k) = c = const$ (chẳng hạn c=1 như băm tuyến tính),* thì *$G(j) = j*c$.*

  Khi đó, *tình trạng "gom tụ" dễ xảy ra khi gặp đụng độ (vì hai khóa khác nhau khi bị đụng độ, việc tìm vị trí trống kế tiếp sẽ được thực hiện theo các bước nhảy giống nhau).*

- Vậy để *<u>tránh hiện tượng gom tụ</u>*, trên *thực tế*, nên chọn H_2 khác H và H_2 phụ thuộc hiển vào khóa. Chẳng hạn:

  $H_2(k) = M' - k \bmod (M')$ (∈ [1, M'])

  hoặc: **$H_2(k) = 1 + k \bmod (M')$** (∈ [1, M']), với *1<M'<M*, ***(6)***

  Với M'=M-1 thì: **$H_2(k) = M-1 - k \bmod (M-1)$** (∈ [1, M-1] ∩ **N**) ***(6')***

  Với M'=M-2 thì: **$H_2(k) = 1 + k \bmod (M-2)$** (∈ [1, M-2] ∩ **N**) ***(6'')***

  Vậy điều kiện (1) được thỏa nếu: ***M nguyên tố, $H_2(k)$ được chọn theo (6) hoặc (6'), hoặc (6'').***

- Để *<u>giảm thời gian tính toán</u>* cho hàm băm thứ 2 này (và do đó giảm thời gian tìm kiếm và các thao tác có liên quan khác trên bảng băm, nhưng khi đó, *điều kiện (1) không luôn thỏa mãn !)*, ta có thể xét *dạng đơn giản hơn chỉ dùng 3 bits cuối của khoá k*:

  $H_2(k) = 8 - k \bmod 8$ (∈ [1, 8]) ***(7)***

* *<u>Ví dụ 6</u>*: Xét dãy các khóa và trị các hàm băm tương ứng được lần lượt đưa vào bảng băm với kích thước M = 19 như sau:

Khóa 1:	A	S	E	**<u>A</u>**	R	C	**<u>H</u>**	I	N	G	**<u>E</u>**	**<u>X</u>**	**<u>A</u>**	**<u>M</u>**	***<u>P</u>***	L	E
Hash 1:	1	0	5	1	18	3	8	9	14	7	5	5	1	13	*16*	12	5
Hash 2:	7	5	3	7	6	5	8	7	2	1	3	8	7	3	*8*	4	3

(với: $H_2(k) = 8 - k \bmod 8$)

Sau khi chèn, ta có bảng *(H.2)*.

* *<u>Nhận xét</u>*:

- Về mặt trung bình, *phương pháp băm kép dùng ít phép thử hơn phương pháp băm thử tuyến tính*:

  . ***Số lần so sánh trung bình*** trong trường hợp *tìm kiếm không thành công* là: C3 = 1/(1 –a) *(< C1. Tại sao ?)*

  . Số lần so sánh trung bình trong trường hợp *tìm kiếm thành công* là:

  C4 = - ln(1-a)/a *(< C2. Tại sao ?)*

HT	***A***	***S***	***E***	***A***	***R***	***C***	***H***	***I***	***N***	***G***	***E***	***X***	***A***	***M***	***P***	***L***	***E***
0	~	**S**	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
1	**A**	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
2	~	~	~	~	~	~	~	~	~	~	~	~	~	~	**P**	P	P
3	~	~	~	~	~	**C**	C	C	C	C	C	C	C	C	C	C	C
4	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~
5	~	~	**E**	E	E	E	E	E	E	E	E	E	E	E	E	E	E
6	~	~	~	~	~	~	~	~	~	~	~	~	~	**M**	M	M	M
7	~	~	~	~	~	~	~	~	~	**G**	G	G	G	G	G	G	G
8	~	~	~	**A**	A	A	A	A	A	A	A	A	A	A	A	A	A
9	~	~	~	~	~	~	~	**I**	I	I	I	I	I	I	I	I	I
10	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~
11	~	~	~	~	~	~	~	~	~	~	**E**	E	E	E	E	E	E
12	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	**L**	L
13	~	~	~	~	~	~	~	~	~	~	~	**X**	X	X	X	X	X
14	~	~	~	~	~	~	~	~	**N**	N	N	N	N	N	N	N	N
15	~	~	~	~	~	~	~	~	~	~	~	~	**A**	A	A	A	A
16	~	~	~	~	~	~	**H**	H	H	H	H	H	H	H	H	H	H
17	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	**E**
18	~	~	~	~	**R**	R	R	R	R	R	R	R	R	R	R	R	R

(H.2)

A	E2	E1
0,10	***1,05***	***1,06***
0,30	1,19	1,21
0,40	1,28	1,33
0,50	*1,39*	*1,50*
0,60	1,53	1,75
0,70	1,72	2,17
0,80	2,01	3,00
0,90	***2,56***	***5,50***
0,95	*3,15*	*10,50*
0,99	***4,65***	***50,50***

(H.3)

- Trong thực tế: số phép thử trung bình *nhỏ hơn 5 lần* cho trường hợp tìm kiếm *không thành công* nếu bảng băm chứa *ít hơn 80%, và* cho trường hợp tìm kiếm *thành công* nếu bảng băm chứa *ít hơn 99% !*
- Tuy vậy, các *phương pháp địa chỉ mở* có thể *bất lợi* trong tình huống *biến động*, khi *số lần thêm vào* và *loại bỏ chưa biết trước*.

Tóm lại, với các *phương pháp biến đổi khóa:*

- Trung bình các phép thử để truy xuất và thêm một khóa (phụ thuộc vào hệ số tải a = n/M) vào bảng băm theo phương pháp:
  . băm thử tuyến tính là: E1 = (1 – a/2) / (1 – a)
  . băm kép là: E2 = - ln(1-a)/a
- Bảng trong *(H.3)* cho ta vài giá trị của E1 và E2 theo giá trị của a. Từ đó, ta thấy các phương pháp biến đổi khóa từ phương pháp thử tuyến tính dù chưa thật tốt đến phương pháp băm kép đều là các *phương pháp tìm kiếm rất hiệu quả.*

- *Tuy vậy**, **nhược điểm** lớn của phương pháp biến đổi khóa là:*

. Kích thước của bảng băm cố định và không thể điều chỉnh theo yêu cầu thực tế. Trong thực tế, ta nên ước lượng trước số phần tử và chọn kích thước bảng băm lớn hơn 10% (khi đó, hệ số tải a ≤ 91%).

. Việc loại bỏ một phần tử khỏi bảng băm trong một số trường hợp (chẳng hạn với phương pháp thử theo địa chỉ mở) khá phức tạp. Để *đơn giản hóa việc xóa (về mặt lô-gic) một phần tử* và *có thể phục hồi lại chúng,* ta có thể *thêm một trường đánh dấu xóa vào mỗi khóa.*

PHỤ LỤC

Các thao tác sơ cấp trên tập tin trong C++

Ta xét hai kiểu tập tin trong ngôn ngữ C++: tập tin *nhị phân* và tập tin *văn bản*.

- **Tập tin nhị phân**: *dữ liệu ghi trên* tập tin *theo các bytes nhị phân giống như khi chúng được lưu trữ ở bộ nhớ trong và chúng không bị biến đổi trong quá trình nhập xuất*. Khi đọc đến cuối tập tin ta nhận được *mã kết thúc tập tin EOF*.

- **Tập tin văn bản**: các tập tin này *lưu trữ các từ theo từng dòng*. Nó *khác tập tin kiểu nhị phân khi xử lý ký tự chuyển dòng và ký tự kết thúc tập tin văn bản* trong các thao tác *đọc và ghi.*

C++ là một trong những ngôn ngữ phục vụ cho phương pháp lập trình hướng đối tượng. Trong phương pháp này, một trong những khái niệm quan trọng là lớp. Có thể xem lớp là công cụ để lưu trữ các đối tượng thông qua cấu trúc dữ liệu để biểu diễn chúng và cả những phương thức cơ bản thao tác trên chúng. Khi làm việc với tập tin trong C++, các thao tác trên tập tin là các phương thức thuộc các lớp *ifstream, ofstream, fstream, ios*.

A/. *Các **phương thức dùng chung cho cả hai kiểu*** tập tin *văn bản và nhị phân*

1) **Mở tập tin.**

* Trước khi làm việc với tập tin (đọc hay ghi) ta phải mở nó để nhận một tên ngoài (tên file thực tế nằm trên đĩa), thực hiện một số việc kiểm tra và phân tích cú pháp, trao đổi với hệ điều hành rồi *tạo ra một tên nội bộ đại diện (biến file hình thức) để dùng về sau trong việc đọc hay ghi lên* tập tin. *Tên nội bộ này là một con trỏ (gọi là con trỏ tập tin)*, trỏ tới cấu trúc chứa thông tin tập tin, chẳng hạn như: vị trí bộ đệm file, vị trí byte hiện tại trong bộ đệm, tập tin đang đọc hay ghi, nối thêm,...

* ***Khai báo*** và ***mở tập tin*** theo cú pháp sau:

fstream BienFileHinhThuc(const char *FileName, int mode);

trong đó *FileName* là tên tập tin có kiểu hằng xâu ký tự, *mode* nhận một số trong các giá trị sau (và chúng nối kết nhau bằng toán tử logic trên bit **¦)**:

ios::in: mở một tập tin *để đọc*. Nếu tập tin *chưa tồn tại sẽ bị lỗi*. Phần chọn này có thể bỏ qua nếu thay lớp *fstream* bởi *ifstream*.

ios::out: mở một tập tin *để ghi mới*. *Nếu tập tin đã tồn tại thì nó bị xóa*. Phần chọn này có thể bỏ qua nếu thay lớp *fstream* bởi *ofstream*.

ios::app: mở một tập tin *để ghi bổ sung*. Nếu tập tin *chưa tồn tại* thì *tạo tập tin mới*.

ios::binary: mở một tập tin theo *kiểu nhị phân. Nếu không có* phần này thì tập tin được mở theo *kiểu văn bản*.

.....

* *Chú ý*:

- Nếu mở một tập tin chưa tồn tại để ghi hay nối thêm thì tập tin sẽ được tạo ra.
- *Mở một tập tin đã có để ghi mới, làm cho nội dung cũ sẽ bị mất* trước khi ghi mới!
- Mở một tập tin chưa có để đọc sẽ sinh lỗi.
- *Nếu có lỗi khi mở tập tin thì BienFileHinhThuc* sẽ nhận giá trị *NULL*.

* *Ví dụ*: Khai báo

```
        char TenFile[] = "Tam.dat";
// mở file nhị phân để đọc và ghi:
        fstream f(TenFile, ios::in ¦ios::out ¦ ios::binary);
/* tương đương với:
        fstream f;
        f.open(TenFile, ios::in ¦ios::out ¦ ios::binary);
*/
        if (!f) cout <<"\nLoi mo file " << TenFile << " de doc va ghi !";
```

có tác dụng mở file TenFile theo kiểu nhị phân, vừa để đọc và để ghi và kiểm tra việc mở file có tốt không.

```
// mở file văn bản để đọc:
        ifsream f(TenFile); tương đương với:
/* tương đương với:
        fstream f;
        f.open(TenFile, ios::in);
*/
```

2) **Đóng tập tin.**

Sau khi mở tập tin và làm các thao tác đọc ghi xong, ta phải *đóng tập tin* theo cú pháp:

BienFileHinhThuc.close();

Phương thức này phá vỡ mối liên hệ giữa *BienFileHinhThuc* và tên ngoài đã được thiết lập. Ngoài ra, nó còn có tác dụng *đẩy nốt nội dung bộ đệm ra file (an toàn dữ liệu)*.

3) **Kiểm tra cuối tập tin.**

BienFileHinhThuc.eof ();

Hàm cho giá *trị khác 0 nếu gặp cuối tập tin* khi đọc, trái lại hàm cho trị 0 (ta thường dùng phương thức này để kiểm tra cuối tập tin sau lệnh đọc sẽ trình bày ở phần sau).

4) **Kiểm tra trạng thái đọc, ghi dữ liệu**:

BienFileHinhThuc.good();

Hàm này trả về 0 nếu gặp lỗi đọc hay ghi và một giá trị khác không trong trường hợp ngược lại.

B/. *Các* ***phương thức dùng cho tập tin kiểu văn bản***

1/ **Đọc 1 chuỗi ký tự**:

char *BienFileHinhThuc.getline(char *line, int maxLine, char delim);

Hàm này *đọc một dòng* (*kể cả dấu xuống dòng và các khoảng trắng*) từ tập tin được chỉ định bởi *BienFileHinhThuc* vào chuỗi ký tự *line*, *nhiều nhất là maxLine-1 ký tự được đọc vào; việc đọc sẽ kết thúc nếu gặp ký tự delim. Dòng kết qủa sẽ được kết thúc bởi '\0'. Thông thường hàm này trả về địa chỉ chuỗi line, trừ khi gặp cuối tập tin hoặc gặp lỗi nó sẽ cho trị NULL.*

Để kiểm tra kết quả việc đọc có thể dùng thêm phương thức:

int BienFileHinhThuc.gcount() trả về số ký tự vừa đọc được.

2/ **Ghi 1 chuỗi ký tự**:

BienFileHinhThuc << Chuỗi;

Toán tử này xuất *Chuỗi* ra file được chỉ định bởi *BienFileHinhThuc.*

3/ **Ghi 1 ký tự.**

BienFileHinhThuc.put(char c);

Hàm *ghi một ký tự* ra file được chỉ định bởi *BienFileHinhThuc*.

4) **Đọc 1 ký tự.**

char BienFileHinhThuc.get();

* Hàm này *đọc một ký tự từ file* được chỉ định bởi *BienFileHinhThuc* và *làm dời chỗ vị trí con trỏ định vị việc đọc trong tập tin đến vị trí kế tiếp*.

* *Hàm get trả về ký tự đọc được, nếu thành công.*

C/. *Các* ***phương thức dùng cho tập tin kiểu nhị phân***

1/ **Ghi một số bytes**:

BienFileHinhThuc.write(const char *buf, int size);

Hàm này ghi vào file được chỉ định bởi *BienFileHinhThuc* một số *size* bytes, bắt đầu từ địa chỉ *buf*. Để kiểm tra kết quả việc đọc có thể dùng thêm phương thức BienFileHinhThuc.good().

2/ **Đọc một số bytes**:

BienFileHinhThuc.read(char *buf, int size);

Hàm này đọc từ file được chỉ định bởi *BienFileHinhThuc* một số *size* bytes và gán chúng vào mảng các ký tự được xác định bởi *buf*.

Có thể dùng phương thức *int BienFileHinhThuc.gcount()* để biết số bytes vừa đọc được.

3/ **Chuyển con trỏ định vị việc ghi trong file**:

BienFileHinhThuc.seekp(long offset, int origin);

Để *truy cập ngẫu nhiên tập tin khi ghi* ta dùng hàm này để đặt con trỏ định vị việc *ghi* trong tập tin được chỉ định bởi *BienFileHinhThuc* di chuyển *offset* bytes từ vị trí xuất phát được xác định theo một trong các giá trị sau của *origin*:

ios::beg tìm *từ đầu* tập tin
ios::cur tìm từ *vị trí hiện hành*
ios::end tìm *từ cuối* tập tin

Phương thức

long *BienFileHinhThuc.tellp();*

trả về vị trí hiện hành của con trỏ định vị việc *ghi* trong tập tin.

4/ **Chuyển con trỏ định vị việc đọc trong file**:

BienFileHinhThuc.seekg(long offset, int origin);

Để *truy cập ngẫu nhiên tập tin khi đọc* ta dùng hàm này để đặt con trỏ định vị việc *đọc* trong tập tin được chỉ định bởi *BienFileHinhThuc* di chuyển *offset* bytes từ vị trí xuất phát được xác định theo giá trị của *origin*.

Phương thức

long *BienFileHinhThuc.tellg();*

trả về vị trí hiện hành của con trỏ định vị việc *đọc* trong tập tin.

Ví dụ: *Kiểm tra số bytes* của một file có tên TenFile đã tồn tại:

```
ifstream f(TenFile);
        f.seekg(0, ios::end);
long SoBytes = f.tellg(); //…
        f.close();
```

Lưu ý khi *truy cập ngẫu nhiên tập tin để đọc hay ghi, các bytes được đánh số bắt đầu từ 0.*

BÀI TẬP
"CẤU TRÚC DỮ LIỆU & GIẢI THUẬT 2"

Các bài tập có đánh dấu (*) là các bài tập khó hoặc cần nhiều thời gian để thực hiện dành cho các học viên khá giỏi. Có thể kết hợp nhiều bài tập (*) có liên quan để hình thành tiểu luận của môn học.

Bài tập chương 1 (File)

(*File có cấu trúc*)

1) Giả sử ta cần lưu danh sách các học viên với số lượng chưa biết trước vào một file "DuLieu.Dat". Mỗi mẫu tin học viên cần lưu các thông tin sau:
- MãSố (kiểu chuỗi ký tự)
- HọTên (kiểu chuỗi ký tự)
- NămSinh (kiểu nguyên)
- Điểm các môn: Toán, Lý, Hóa(kiểu thực)
- PhânLoại (kiểu ký tự: A,B,C,D phụ thuộc điểm trung bình của 3 môn trên).

Hãy lập chương trình có các chức năng sau:

a. *Thêm một mẫu tin vào cuối file.*

b. *Tạo file với dữ liệu lấy từ bàn phím* hoặc từ một file khác có cùng cấu trúc (*sao chép file*) hoặc được tạo tự động một cách ngẫu nhiên khi số lượng các các mẫu tin cần nhập khá lớn.

c. *Khai thác file theo một tiêu chuẩn nào đó* mà không thay đổi file nguồn, chẳng hạn:
- Tìm mẫu tin đầu tiên (bắt đầu từ mẫu tin tại vị trí xác định) thỏa một tính chất nào đó. Trong trường hợp tìm thấy, hãy trả về số thứ tự của mẫu tin vừa tìm thấy trong file.
- Tìm và xuất các mẫu tin trên file thỏa một tính chất nào đó trong hai trường hợp:

  . tính chất chỉ phụ thuộc vào một thuộc tính của mẫu tin (ví dụ: mẫu tin có trường HọTên trùng với một chuỗi ký tự cho trước).

  . tính chất phụ thuộc vào nhiều thuộc tính của mẫu tin (ví dụ: mẫu tin có: trường HọTên trùng với một chuỗi ký tự cho trước và có điểm trung bình lớn hơn 5.0).

d. Cập nhật các mẫu tin của file nguồn (và có thể làm thay đổi file nguồn, có thể sử dụng hoặc không sử dụng các file phụ), chẳng hạn:
- *Sửa một mẫu tin đầu tiên* trong file nguồn thỏa một tính chất nào đó.
- *Sửa mọi mẫu tin* trong file nguồn thỏa một tính chất nào đó.
- *Chèn* một mẫu tin mới vào sau mẫu tin đầu tiên trong file nguồn thỏa một tính chất nào đó.
- *Xoá mẫu tin đầu tiên* trong file nguồn thỏa một tính chất nào đó.

- *Xoá mọi mẫu tin* trong file nguồn thỏa một tính chất nào đó (khi xóa chú ý có thể: chỉ đánh dấu xóa hoặc xóa hẳn mẫu tin tìm thấy).

e. (*) Sắp xếp file tăng dần theo một tiêu chuẩn nào đó bằng một trong các phương pháp:
   - chèn
   - trộn: trộn trực tiếp, *trộn tự nhiên*, trộn nhiều đường cân bằng (*)

   Kiểm tra lại các hàm sắp xếp trên với bộ dữ liệu sau:

   66 31 22 97 36 15 6 32 79 44
   26 19 45 46 75 8 13 17 62 88
   76 33 72

   Yêu cầu thực hiện các thao tác cơ bản trên đây khi file được tổ chức theo kiểu:
   i) *tuần tự (cho phép hoặc không cho phép đánh dấu xóa)*
   ii) chỉ mục (**).

2) (**) Viết các thao tác cơ bản (tìm kiếm, chèn, xóa, sửa, sắp xếp theo các quan hệ thứ tự khác nhau,…) trên file được tổ chức như kiểu cây BST hay cây cân bằng AVL theo hai cách:

i) tổ chức chung vào một file chứa dữ liệu gốc cùng các trường liên kết chứa địa chỉ các nút con trái và phải;

ii) ngoài file dữ liệu gốc f, tổ chức thêm các file chỉ mục f_idx chỉ chứa các trường liên kết chứa địa chỉ các nút con trái và phải trong f theo những quan hệ thứ tự khác nhau phát sinh động trong thực tế.

(File văn bản)

3) Giả sử ta cần lưu trữ các chuỗi ký tự theo từng dòng vào một file văn bản "DữLiệu.Txt" và khai thác thông tin trên file này. Hãy lập chương trình có các chức năng sau:

a. *Lưu vào file văn bản "DữLiệu.Txt" các dòng chuỗi ký tự được lấy từ bàn phím* hoặc từ một file văn bản cho trước.

b. *Khai thác file văn bản theo một tiêu chuẩn nào đó mà không làm thay đổi file nguồn (các thông tin thỏa tiêu chuẩn được kết xuất ra màn hình* hoặc một file khác). Chẳng hạn: tính tần số xuất hiện của các từ hay dòng trong file thỏa một tính chất nào đó; xuất ra (màn hình theo từng trang 23 dòng) các từ hay dòng của file nguồn thỏa một tính chất nào đó; …

c. Sao chép file "DữLiệu.Txt" vào một file văn bản khác .

d. Nối hai file văn bản f1.txt và f2.txt cho trước trong hai trường hợp:
   - file f2.txt được nối vào cuối file f1.txt (file f1.txt có thể thay đổi).
   - tạo file f3.txt kết qủa mới, còn hai file đầu không bị thay đổi.

e. (*) Chia file văn bản nguồn thành nhiều file văn bản con thỏa hay không thỏa một tính chất nào đó. Chẳng hạn tách file văn bản chương trình nguồn trong C++ thành hai file con: một file chứa tất cả các chú thích trên dòng và

trên đoạn (giả sử các chú thích không lồng nhau), file còn lại không chứa các chú thích.

f. (*) Cập nhật thông tin của file văn bản nguồn (và có thể làm thay đổi file nguồn, có thể sử dụng hoặc không sử dụng các file phụ), chẳng hạn:
- Thay thế (hay sửa) các đơn vị (ký tự, từ hoặc dòng) của file văn bản nguồn bởi các đơn vị xác định khác.
- Xóa các đơn vị (ký tự, từ hoặc dòng) của file văn bản nguồn bởi các đơn vị xác định khác.
- Chèn một đơn vị (ký tự, từ hoặc dòng) xác định vào sau những đơn vị trong file nguồn thỏa một tính chất nào đó.

4) a. Tạo ra các file văn bản *mt.txt, day.txt* và *dt.txt* chứa các giá trị của:

α. ma trận cấp n có dạng sau:

```
3
12      32      -4
-1      45      9
27      -87     34
```

trong đó: dòng đầu lưu cấp *n* của ma trận, các dòng kế tiếp lưu các dòng tương ứng của ma trận.

β. dãy gồm *n* số thực có dạng sau:

```
5
1.3     5.67    7.23    0.12    -9.67
```

trong đó: dòng đầu lưu số phần tử *n* của dãy, dòng kế tiếp lưu các phần tử tương ứng của dãy.

γ. Dãy *n* đường thẳng trong mặt phẳng có dạng sau:

```
2
1.3     5.67    7.23    0.12
-9.67   12      32      -4
```

trong đó: dòng đầu lưu số đường thẳng *n* của dãy, các dòng kế tiếp lưu các cặp tọa độ của hai điểm (x_1, y_1), (x_2, y_2) trên từng đường thẳng tương ứng của dãy.

b. Đọc nội dung của các file văn bản trên, kiểm tra tính hợp lệ của dữ liệu tương ứng và:

α. Kiểm tra nó có phải là ma phương hay không ? (tổng mỗi dòng, mỗi cột và mỗi đường chéo của ma trận đều bằng nhau).

β. - Tính tổng các giá trị âm và tìm giá trị lớn nhất trong các phần tử của dãy.
- (*) Tìm 3 giá trị lớn nhất trong các phần tử của dãy.

γ. Viết phương trình các đường thẳng đã cho (trong mặt phẳng).

(File nhị phân các bytes hay ký tự)

5) (*)

a) Tạo ra 2 file nhị phân chứa các ký tự trong các trường hợp sau:

- mọi ký tự đều có mã ASCII lớn hơn 0x1F (‘\31’).
- mọi ký tự đều có mã ASCII hoặc lớn hơn 0x1F (‘\31’), hoặc bằng 0x1A (chú ý trường hợp ở giữa file có chứa ký tự 0x1A (‘\26’) để kết thúc file văn bản).
- các ký tự có mã ASCII bất kỳ.

b) Hỏi trong trường hợp nào có thể mở file ra đọc theo kiểu nhị phân, sau đó theo kiểu văn bản và xuất được nội dung vừa đọc ra màn hình ? So sánh và cho nhận xét.

c) *Sao chép hai file bất kỳ* (nội dung và kích thước hai file giống hệt nhau).

d) Chia một file bất kỳ thành n file con có kích thước bằng nhau (trừ file cuối cùng).

e) Nối n file bất kỳ thành một file mới có kích thước đúng bằng tổng kích thước của n file đã cho.

f) (*) Mã hóa và giải mã một file bất kỳ theo một phương pháp mã đơn giản nào đó.

Bài tập chương 2 (B - cây)

1) a. *Xây dựng B-cây cấp 2* từ các mục dữ liệu đầu vào như sau (trình bày từng bước khi chèn mỗi phần tử):

i) 25, 17, 31, 42, 21, 19, 26, 33, 47, 44, 45, 43, 8, 9;

ii) 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1;

iii) ca, ea, ba, da, bf, df, ah, cg, bi, cc, af, eg, bd, ec, ch, ai, dc, di, ce, ef, cf.

b. *Hãy trình bày từng bước của B-cây cấp 2 ở câu a.iii) khi xóa lần lượt các phần tử* sau: cf, ef, ce, di, dc, ai, ch, eg, af, cc, bi, cg, ah, df, bf.

2) (*) Viết chương trình cài đặt B-cây cấp n ở bộ nhớ trong có các chức năng sau:

a. Tìm và chèn một phần tử vào B-cây cấp n.
b. Tạo B-cây cấp n với dữ liệu lấy từ bàn phím.
c. Tìm và xóa một phần tử khỏi B-cây cấp n.
d. Hủy B- cây cấp n.

Dùng chương trình để kiểm tra lại các bộ số liệu ở bài tập 1) trên đây.

3) (**) Tương tự như bài 2 nhưng B-cây cấp n được cài đặt ở bộ nhớ ngoài.

Bài tập chương 3 (Bảng băm)

1) Hãy *tạo ra bảng băm* khi lần lượt chèn từng phần tử từ các bộ số liệu đầu vào sau:

534 702 105 523 959 699 821 883 842 686
658 4 20 382 570 344

khi dùng phương pháp:

a. *Băm liên kết trực tiếp* (dựa trên mảng hoặc danh sách liên kết) trong hai trường hợp: *M=13,* M=17.
b. *Băm liên kết kết hợp* khi M = 17.
c. *Băm theo địa chỉ mở* khi M = 19:
   - Phương pháp *băm thử tuyến tính.*
   - Phương pháp băm thử bậc hai.
   - Phương pháp *băm kép* (dùng hàm băm thứ 2 là: $H_2(k) = 1 + k \bmod (M-2)$).

Trong mỗi trường hợp, tìm số trung bình các phép so sánh khi tìm kiếm một khóa trong bảng băm.

2) *Viết chương trình* để:

a. Tạo bảng băm được cài đặt ở bộ nhớ trong theo các phương pháp đã nêu và kiểm tra lại các bộ dữ liệu đầu vào ở bài tập 1) trên đây.

b. Với mỗi phương pháp băm, viết các thao tác cơ bản trên bảng băm: chèn, tìm kiếm, xóa, phục hồi (khi có thêm trường đánh dấu xóa) các đối tượng dữ liệu, ...

3) (**) Tương tự như bài tập 2 nhưng bảng băm được cài đặt ở bộ nhớ ngoài.

4) (*) Chứng minh các khẳng định lý thuyết ứng với các phương pháp băm theo địa chỉ mở.

5) (*) Cải tiến phương pháp băm liên kết trực tiếp bởi phương pháp *băm liên kết nhị phân*: Dùng mảng băm chứa các nút gốc của:

a. cây tìm kiếm nhị phân BST, hoặc
b. (**) cây cân bằng AVL,

(mỗi cây chứa các khóa có cùng trị băm).

Khi đó viết lại tất cả các thao tác cơ bản trên bảng băm: tìm kiếm, chèn, xóa, sửa các đối tượng, ...

Thể hiện phương pháp băm liên kết ở:

i) bộ nhớ trong.
ii) (**) bộ nhớ ngoài.

TÀI LIỆU THAM KHẢO

[1] A.V. AHO , J.E. HOPCROFT , J.D. ULMANN: *Data structures and algorithms*. Addition Wesley - 1983.

[2] DONALD KNUTH: *The Art of Programming. (vol. 1: Fundamental Algorithms, vol. 3: Sorting and Searching)*. Addition Wesley Puplishing Company - 1973.

[3] ĐINH MẠNH TƯỜNG: *Cấu trúc dữ liệu và thuật toán*. NXB KHKT – 2001.

[4] ĐỖ XUÂN LÔI: *Cấu trúc dữ liệu và giải thuật*. NXB KHKT - 1995.

[5] IAN PARBERRY, *"Lecture Notes on Algorithm Analysis and Computational Complexity (Fourth Edition)"*, Department of Computer Sciences, University of North Texas, December 2001.

[6] LARRY N. HOFF, SANFORD LEESTMA: *Lập trình nâng cao bằng Pascal với các cấu trúc dữ liệu*. Bản dịch của Lê Minh Trung. Công ty Scitec - 1991.

[7] NGUYỄN TRUNG TRỰC: *Cấu trúc dữ liệu*, Trung tâm điện toán, ĐH Bách khoa TP HCM, 1992.

[8] NIKLAUS WIRTH: *Algorithms + Data Structures = Programs*. Prentice - Hall INC - 1976.

[9] ROBERT SEDGEWICK: *Cẩm nang thuật toán, tập 1*, Bản dịch của nhóm tác giả ĐHTH TP HCM, NXB KHKT, 1994.

[10] TRẦN HẠNH NHI, DƯƠNG ANH ĐỨC: *Nhập môn cấu trúc dữ liệu và thuật toán*, Giáo trình của khoa Công nghệ Thông tin, Đại học Khoa học Tự nhiên TP HCM, 2000.