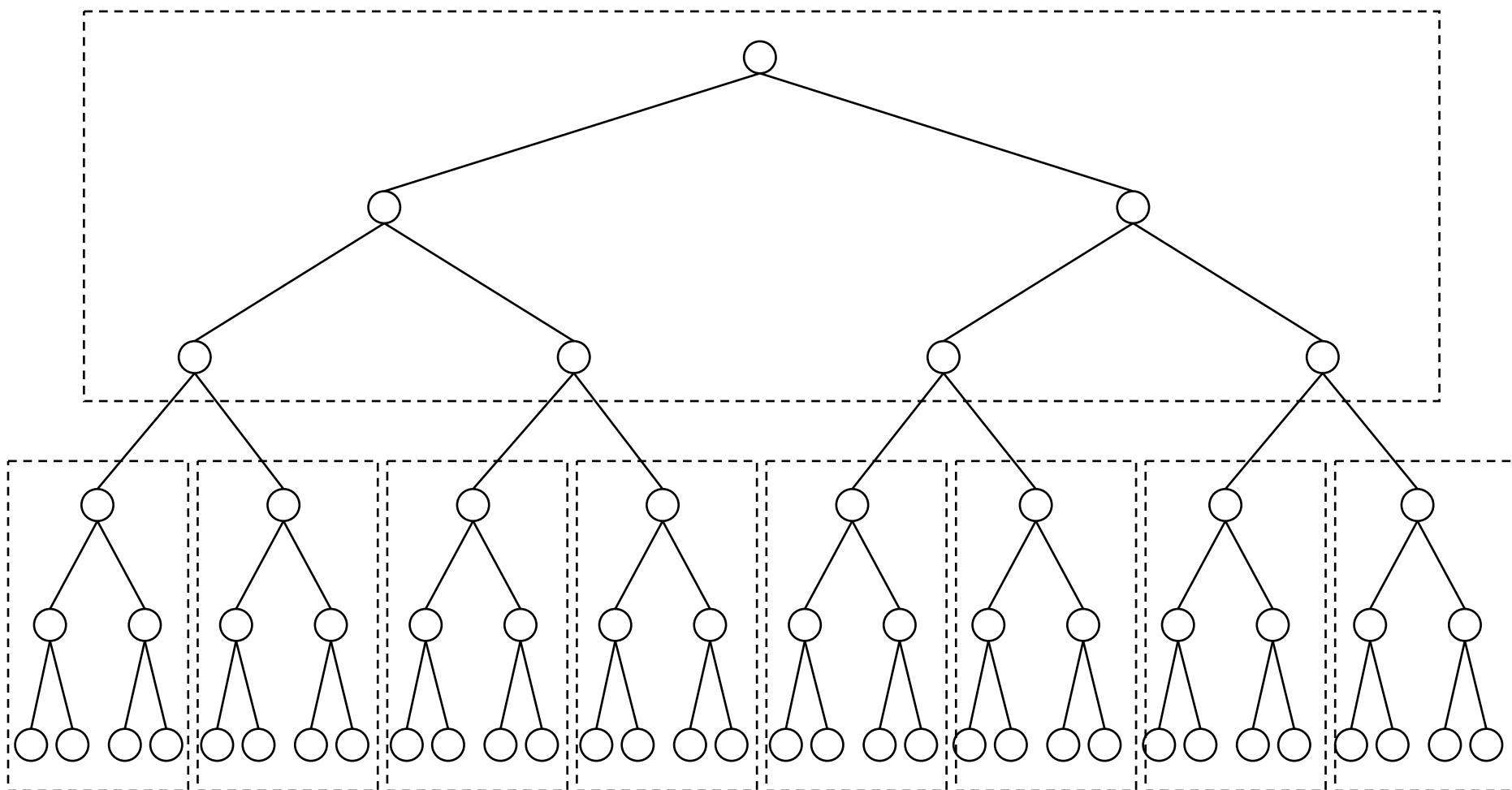


B – cây

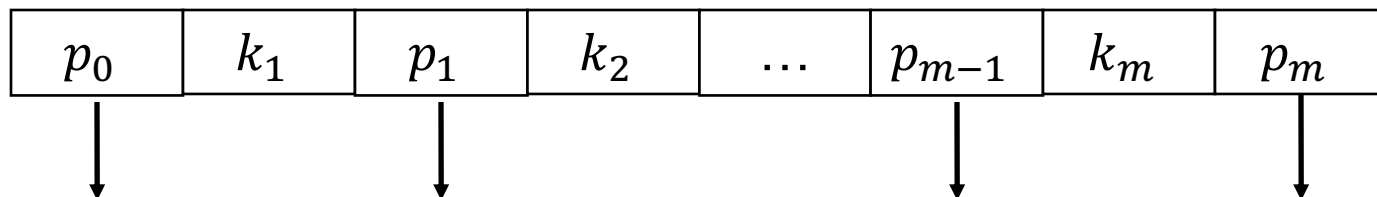
- Cây nhị phân đáp ứng khá đầy đủ các yêu cầu về biểu diễn cấu trúc dữ liệu
- Trong thực tế, kích thước dữ liệu thường là lớn hoặc vô cùng lớn. Đòi hỏi phải lưu trữ ở bộ nhớ ngoài
 - Dung lượng “vô hạn”
 - Tốc độ chậm hơn khoảng 10^5 lần so với bộ nhớ trong, ngoài ra còn là chi phí di chuyển đầu đọc

- Sử dụng *cây nhiều nhánh* (lưu trữ trên đĩa) để biểu diễn cây tìm kiếm cỡ lớn
 - Nhằm hạn chế tối đa thao tác truy xuất (đọc/ghi đĩa), cây được thiết kế với:
 - Đơn vị truy xuất là nhóm dữ liệu, gọi là *trang*
 - Chiều cao cây đạt mức tối thiểu
- Hình thành khái niệm B-cây
- Một trang tương ứng một nút của B-cây
 - Mỗi trang ngoài việc lưu trữ dữ liệu còn chứa “liên kết” để quản lý các trang con

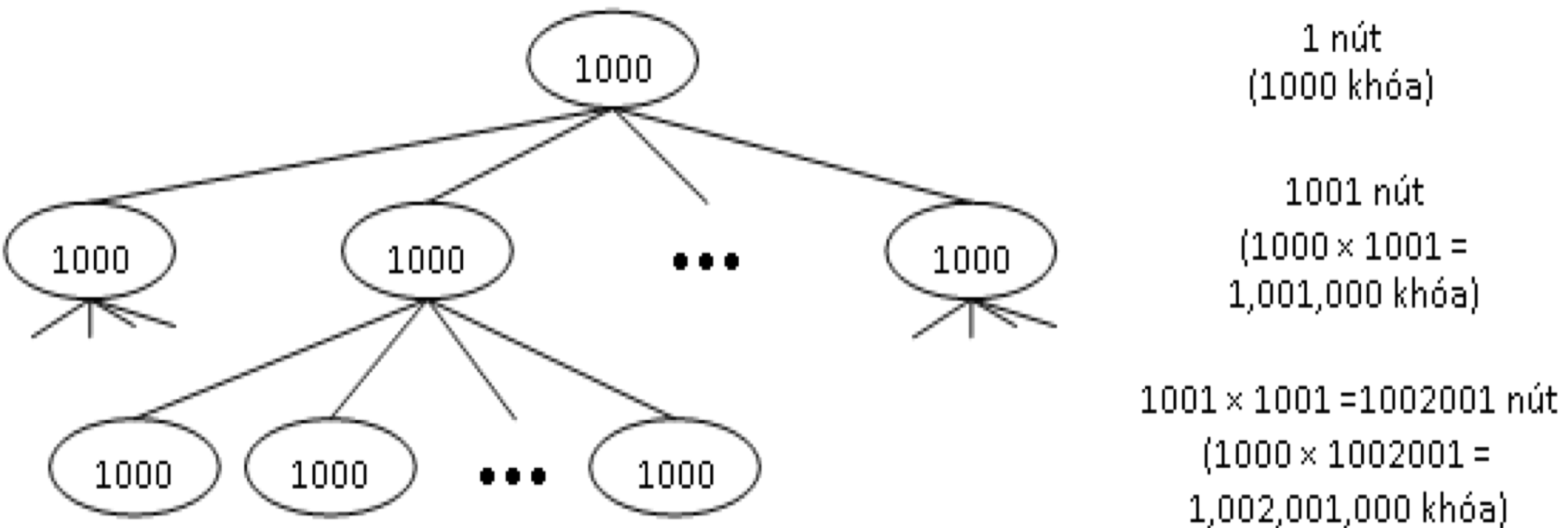


Định nghĩa: B-cây (bậc n) là cây cỡ lớn, được lưu trên đĩa với tổ chức như sau:

- Mỗi trang chứa tối đa $2n$ khóa
- Mỗi trang, trừ trang gốc, chứa tối thiểu n khóa
- Mỗi trang, trừ trang lá, nếu có m khóa thì có $m + 1$ trang con
- Mọi trang lá đều xuất hiện ở cùng mức
- Các khóa được sắp xếp tăng dần từ trái qua phải



Xét B-cây với số khóa trung bình là 1000



- Với chiều cao là 3, cây chứa hơn 1 tỉ khóa (tương ứng với hơn 1 tỉ đơn vị dữ liệu)
- Nút gốc thường được lưu trong bộ nhớ chính nên chỉ mất tối đa 2 phép truy xuất để tìm một khóa

Các cấu trúc dữ liệu

```
const int n = 2;  
const int nn = 4;  
typedef struct page * ref;  
struct item {  
    int    key;  
    ref    p;  
    int    count;  
};  
struct page {  
    int    m;  
    item   e[nn + 1];  
};
```

Tìm kiếm trên B-cây

1. Đọc trang vào bộ nhớ và tìm kiếm tuần tự
 - Nếu m đủ lớn thì dùng tìm kiếm nhị phân
2. Nếu không tìm thấy (gọi k là khóa cần tìm):
 - a) $k_i < k < k_{i+1}$ với $1 \leq i < m$: tìm tiếp trên trang trỏ bởi p_i
 - b) $k_m < k$: tìm tiếp trên trang trỏ bởi p_m
 - c) $k < k_1$: tìm tiếp trên trang trỏ bởi p_0
3. Nếu con trỏ trỏ đến trang mới bằng NULL: không tìm thấy

Thêm phần tử vào B-cây

1. Tìm trang lá thích hợp và chèn vào đúng vị trí
2. Nếu trang đầy: tiến hành tách trang
 - Mỗi trang chứa n khóa, phần tử đứng giữa được đưa lên trang cha
3. Trường hợp trang cha cũng trở nên đầy: Tiến trình tiếp tục theo chiều đi lên
 - Cây lớn lên từ lá đến gốc

Xóa phần tử trên B-cây

Giai đoạn 1: Tìm và xóa phần tử

- ở trang lá: xóa bình thường
- ở trang trong:
 - Tìm *phần tử thay thế* (ở trang lá)
 - Đi theo con trỏ phải nhất của trang con bên trái, hoặc
 - Đi theo con trỏ trái nhất của trang con bên phải
 - Sao chép dữ liệu của *phần tử thay thế* cho phần tử định xóa ban đầu
 - Xóa *phần tử thay thế*

Giai đoạn 2: Kiểm tra “tính cân bằng”

- Việc xóa làm giảm số phần tử m của trang lá đi 1 đơn vị
- Nếu $m \geq n$: Dừng
- Nếu $m < n$: *Xét trang kế bên*
 - $> n$ phần tử: Cân bằng số lượng phần tử của 2 trang (thông qua phần tử trung gian ở trang cha)
 - $= n$ phần tử: Kéo phần tử trung gian ở trang cha xuống và ghép 2 trang \rightarrow trang đầy

Quá trình có thể lan truyền ngược ... (nếu trang cha trở nên thiếu) ... về đến gốc \rightarrow cây giảm chiều cao

```

ref    root, q;
bool   h;
item   u;

...

root = NULL;
for (i = 0; i < n; i++) {
    timthem(a[i], root, h, u);
    if (h) {
        q = root;
        root = new page;
        root->m = 1;
        root->e[0].p = q;
        root->e[1] = u;
    }
}

```

```
void timthem(int x, ref a, bool &h, item &v) {  
    int    r;  
    ref    q;  
    item   u;  
  
    if (a == NULL) {  
        h = true;  
        v.key = x;  
        v.count = 1;  
        v.p = NULL;  
    }  
    ...  
}
```

```

else {
    a->e[0].key = x;  // Lính canh
    for (r = a->m; a->e[r].key > x; r--) ;
    if ((r) && (a->e[r].key == x)) {  // Đã có
        a->e[r].count ++;
        h = false;
    }
    else {
        if (r == 0)      q = a->e[0].p;
        else             q = a->e[r].p;
        timthem(x, q, h, u);
        if (h)           // a trở vào trang lá
            them(r, a, h, u, v);
    }
}
}

```

```

void  them(int r, ref a, bool &h, item &u,
                                     item &v) {

    if (a->m < nn) {    // Trang chưa đầy
        a->m ++;
        h = false;
        for (i = a->m; i >= r + 2; i --)
            a->e[i] = a->e[i - 1];
        a->e[r + 1] = u;
    }
    ...
}

```

```

else {    // Trang trở bởi a đầy
    ref b = new page;
    if (r <= n) {
        if (r == n)
            v = u;
        else {
            v = a->e[n];
            for (i = n; i >= r + 2; i--)
                a->e[i] = a->e[i - 1];
            a->e[r + 1] = u;
        }
        for (i = 1; i <= n; i++)
            b->e[i] = a->e[i + n];
    }
    ...
}

```



```
else {  
    r -= n;  
    v = a->e[n + 1];  
    for (i = 1; i <= r - 1; i++)  
        b->e[i] = a->e[i + n + 1];  
    b->e[r] = u;  
    for (i = r + 1; i <= n; i++)  
        b->e[i] = a->e[i + n];  
}  
  
a->m = b->m = n;  
b->e[0].p = v.p;  
v.p = b;
```

```

ref    root, q;
bool   h;
int    x;

...

cout << "\nNhap vao khoa can xoa: ";
cin >> x;
timxoa(x, root, h);
if (h)
    if (root->m == 0) {
        q = root;
        root = q->e[0].p;
        delete q;
    }

```

...

```

void  timxoa(int x, ref a, bool &h) {
    if (a == NULL)
        h = false;
    else {
        a->e[0].key = x;
        for (r = a->m; a->e[r].key > x; r--)
            ;
        if (r == 0)
            q = a->e[0].p;
        else
            if ((r) && (a->e[r].key == x))
                q = a->e[r - 1].p;
            else
                q = a->e[r].p;
        ...
    } // void

```

```

if ((r) && (a->e[r].key == x))
    if (q == NULL) {          // Trang la
        a->m --;
        h = a->m < n;
        for (i = r; i <= a->m; i++)
            a->e[i] = a->e[i + 1];
    }
    else {
        xoa(a, q, r, h); // Tim phan tu thay the
        if (h) canbang(a, q, r - 1, h);
    }
else {
    timxoa(x, q, h);
    if (h) canbang(a, q, r, h);
}
}

```

```

void xoa(ref a, ref p, int r, bool &h) {
    ref q = p->e[p->m].p;

    if (q) {
        xoa(a, q, r, h);
        if (h)
            canbang(p, q, p->m, h);
    }
    else { // p trở đến trang lá
        p->e[p->m].p = a->e[r].p;
        a->e[r] = p->e[p->m]; // Sao chép
        p->m--;
        h = p->m < n;
    }
}

```

```

void  canbang(ref c, ref a, int s, bool &h)
{
    int mc = c->m;
    if (s < mc) {
        s++;
        b = c->e[s].p;
        mb = b->m;

        k = (mb - n + 1)/2; //Số phần tử chuyển giao
        a->e[n] = c->e[s]; // Lấy phần tử ở trang cha
        a->e[n].p = b->e[0].p;

        ...
    } // void

```

```
if (k > 0) { // Khong can phai ghep trang
    for (i = 1; i <= k - 1; i++)
        a->e[n + i] = b->e[i];
    c->e[s] = b->e[k];
    c->e[s].p = b;
    b->e[0].p = b->e[k].p;
    mb -= k;
    for (i = 1; i <= mb; i++)
        b->e[i] = b->e[i + k];
    b->m = mb;
    a->m = (n - 1) + k;
    h = false;
}
```

```

else { // Trộn trang a và b
    for (i = 1; i <= n; i++)
        a->e[n + i] = b->e[i];
    for (i = s; i <= mc - 1; i++)
        c->e[i] = c->e[i + 1];
    a->m = nn;
    c->m = mc - 1;

    delete b;
}
}
...

```



```
else { // s = mc
    if (s == 1)
        b = c->e[0].p;
    else
        b = c->e[s - 1].p;

    mb = b->m + 1;
    k = (mb - n) / 2; // Số phần tử chuyển giao
    ...
}
```

```

if (k > 0) {
    for (i = n - 1; i >= 1; i--)
        a->e[i + k] = a->e[i];
    a->e[k] = c->e[s];
    a->e[k].p = a->e[0].p;
    mb -= k;
    for (i = k - 1; i >= 1; i--)
        a->e[i] = b->e[i + mb];
    a->e[0].p = b->e[mb].p;
    c->e[s] = b->e[mb];
    c->e[s].p = a;
    b->m = mb - 1;
    a->m = (n - 1) + k;
    h = false;
}

```

...

```
else { // Trộn hai trang a và b
```

```
    b->e[mb] = c->e[s];
```

```
    b->e[mb].p = a->e[0].p;
```

```
    for (i = 1; i <= n - 1; i++)
```

```
        b->e[i + mb] = a->e[i];
```

```
    b->m = nn;
```

```
    c->m = mc - 1;
```

```
    delete a;
```

```
}
```

```
}
```