Slide bài giảng môn Cấu trúc dữ liệu và giải thuật (09/2018)

KIỂU DỮ LIỆU TRỪU TƯỢNG

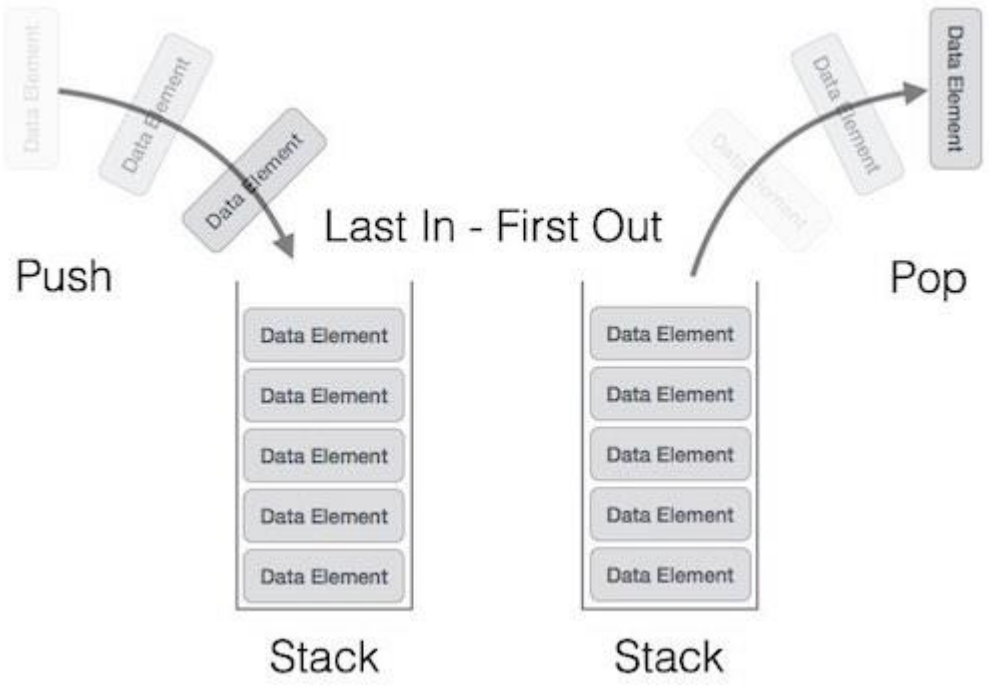
TS. Nguyễn Ngọc Thảo

Bộ môn Khoa học Máy tính, Khoa CNTT, ĐHKHTN

Tp. HCM, 09/2018

Nội dung bài giảng

- Ngăn xếp (stack)
- Hàng đợi (queue)
  - Hàng đợi ưu tiên
- Bảng băm (hash table)
  - Phương pháp xây dựng hàm băm
  - Xử lý đụng độ trong bảng băm

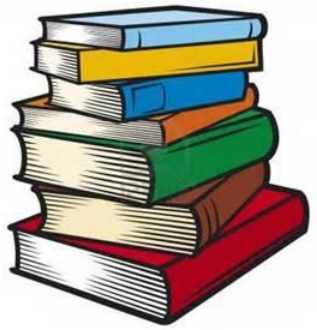


Ngăn xếp (Stack)

Ngăn xếp là gì?



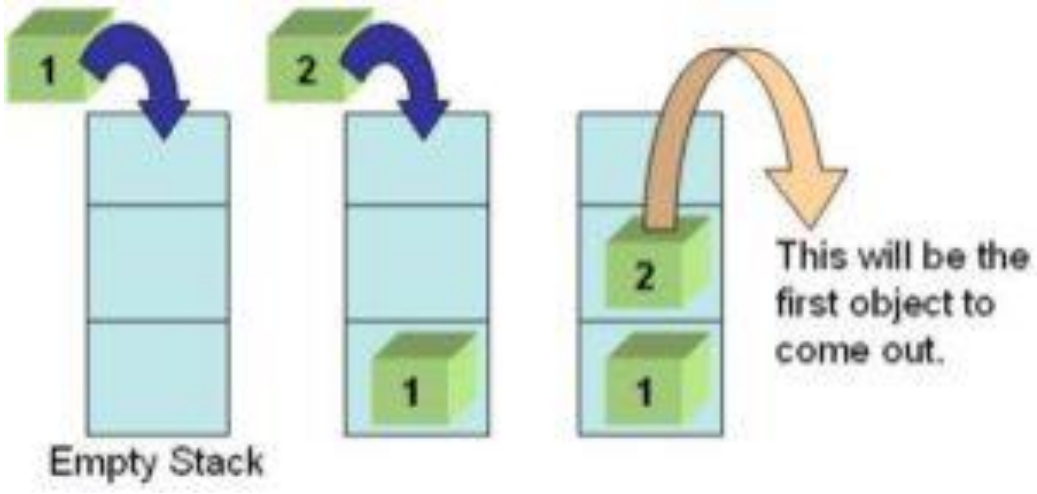




- Ngăn xếp được triển khai phổ biến trong đời thực.
- Chỉ có thể thao tác tại một đầu của ngăn xếp (đỉnh).

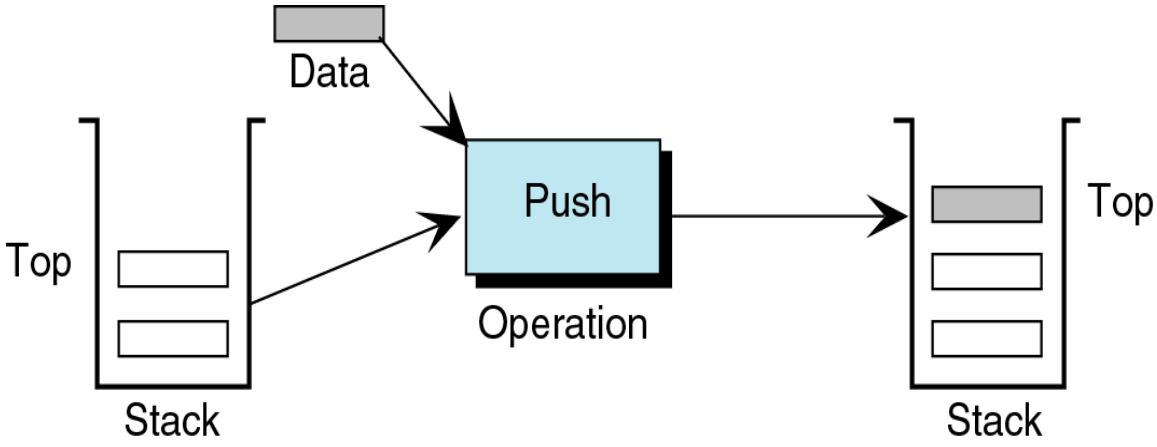
Định nghĩa ngăn xếp

- **Ngăn xếp** là cấu trúc dữ liệu trừu tượng (ADT) được sử dụng trong nhiều ngôn ngữ lập trình.
  - Ví dụ, C++ std::stack, C# Collections.Stack, Java java.util.* Stack
- Các phần tử được tổ chức thứ tự theo cơ chế LIFO.
  - **Last In First Out:** phần tử đặt cuối cùng vào ngăn xếp sẽ được lấy ra đầu tiên.
  - Các phần tử đi vào và lấy ra ở cùng một đầu (top).



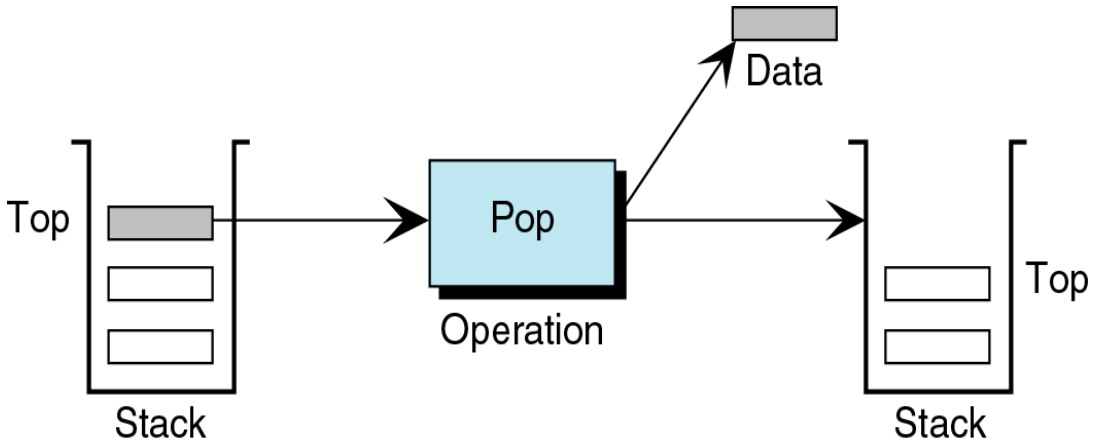
Thao tác cơ bản trên ngăn xếp

- **`isEmpty()`:** kiểm tra tình trạng của ngăn xếp
- **`push(newEntry)`:** thêm một phần tử `newEntry` vào (đỉnh) ngăn xếp, có thể làm ngăn xếp đầy

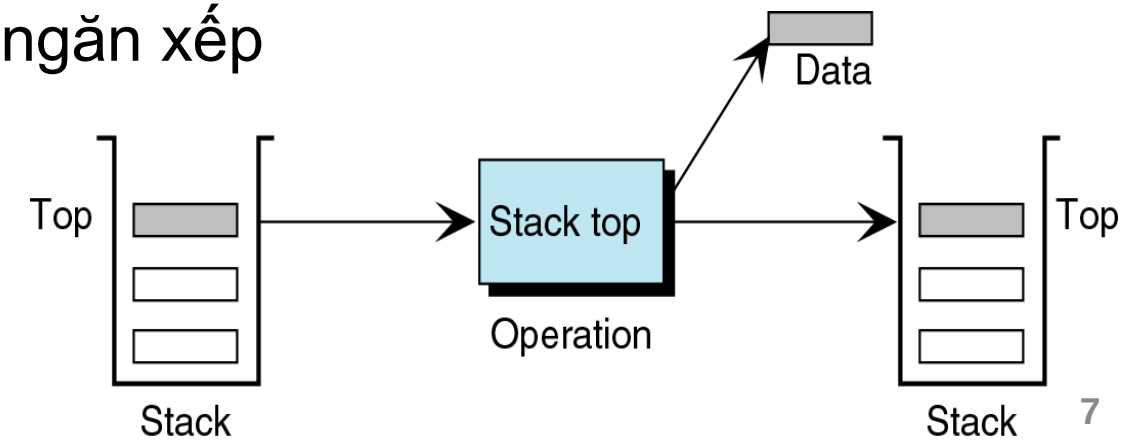


Thao tác cơ bản trên ngăn xếp

- **`pop()`:** lấy một phần tử từ (đỉnh) ngăn xếp, có thể làm ngăn xếp rỗng



- **`peek()`:** xem phần tử ở đỉnh ngăn xếp nhưng không làm thay đổi nội dung của ngăn xếp



Ví dụ áp dụng ngăn xếp

- Cho chuỗi thao tác sau:

EAS*Y**QUE***ST***I*ON

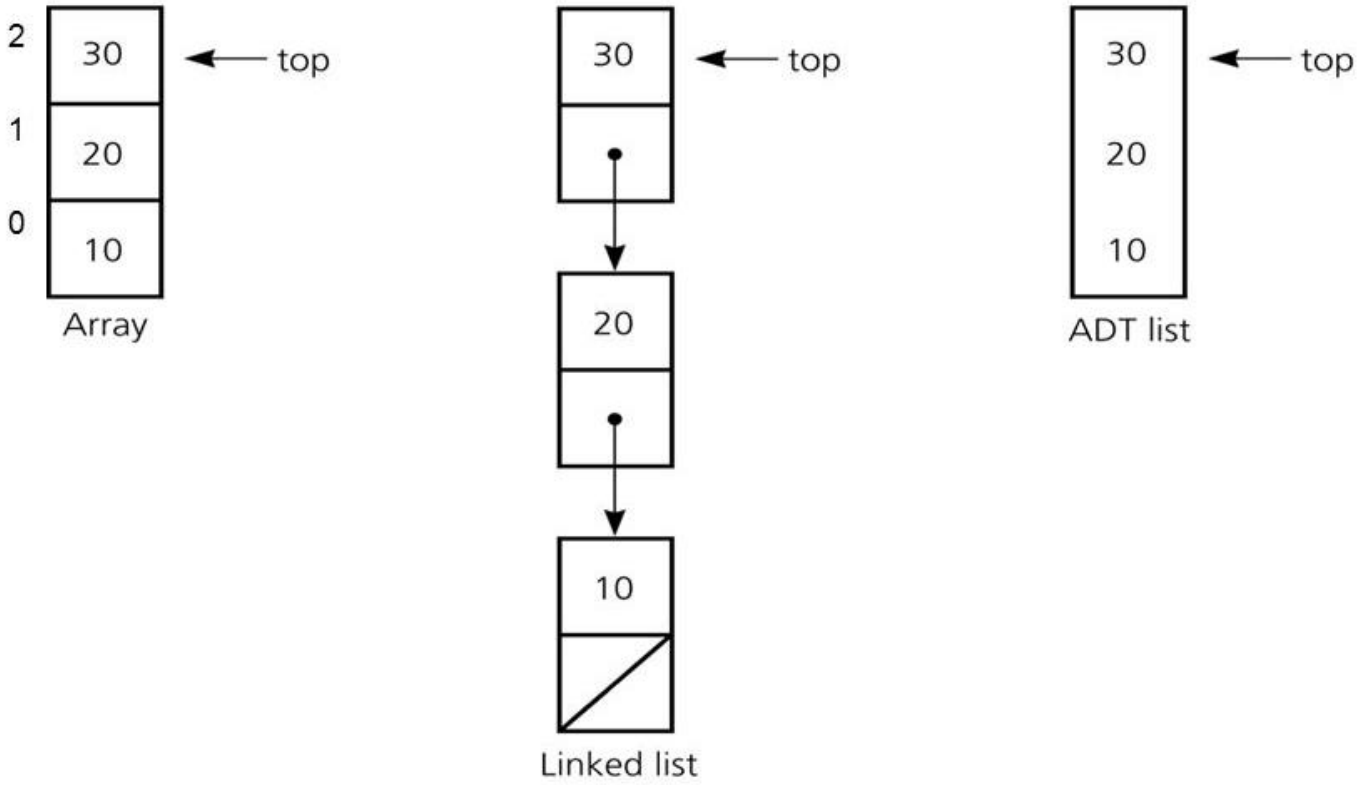
trong đó

- *ký tự*: đẩy ký tự tương ứng vào ngăn xếp
- *: lấy một phần tử trong ngăn xếp và xuất ra màn hình

- Cho biết kết quả cuối cùng trên màn hình?

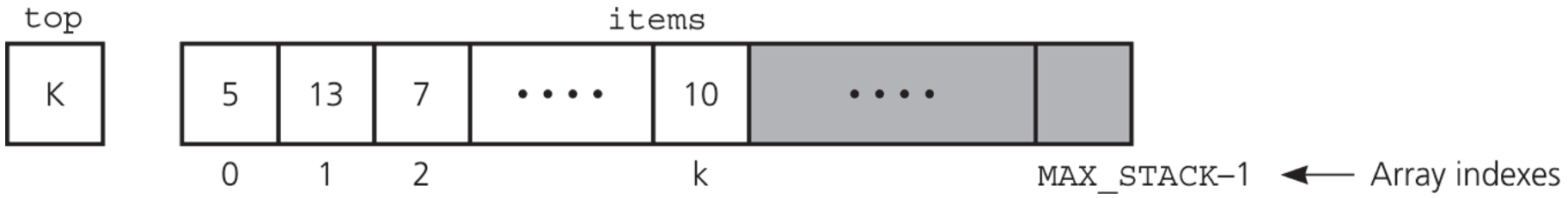
Cài đặt ngăn xếp

- Mảng (ArrayStack)
- Danh sách liên kết (LinkedStack)
- Kiểu dữ liệu danh sách (ADTListStack)



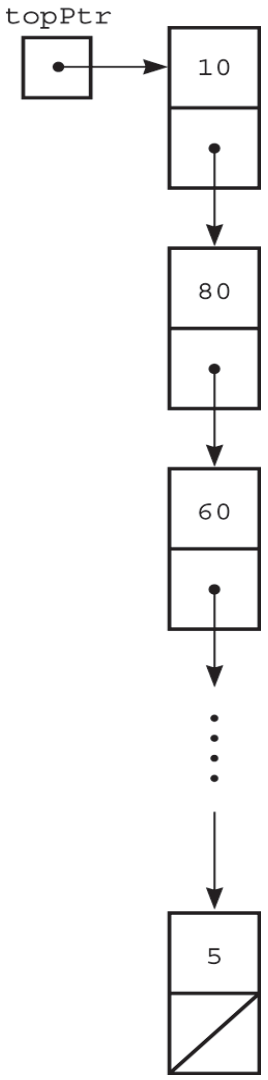
Cài đặt ngăn xếp bằng mảng

```
template<class ItemType>
class ArrayStack{
   // Array of stack items
   ItemType items[MAX_STACK];
   // Index to top of stack
   int      top;
}
```



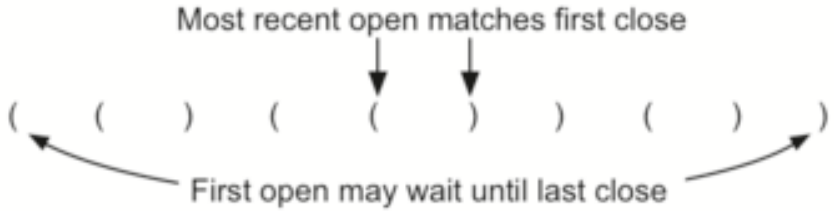
Cài đặt ngăn xếp bằng DSLK

- Sử dụng khi ngăn xếp thay đổi kích thước động
- **`peek()`** tham chiếu đến con trỏ đầu (head) của một danh sách liên kết các phần tử
- Thao tác **`push(newEntry)`** và **`pop()`** thêm một phần tử và xóa phần tử ở đầu danh sách liên kết




```
template<class ItemType>
class LinkedStack{
    // Array of stack items
    Node<ItemType>* topPtr;
    // Pointer to first node in the chain;
    // this node contains the stack's top
}
```

Ứng dụng của ngăn xếp



Kiểm tra dấu ngoặc cân bằng

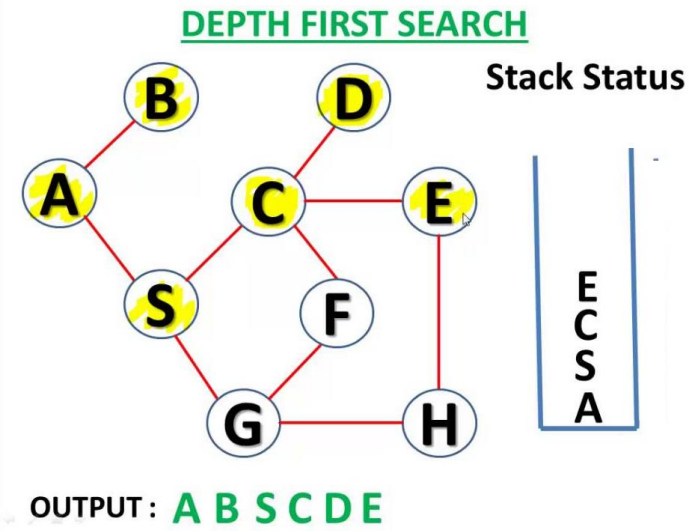
$$50 + 3 * 5 + 7$$

Tính giá trị biểu thức

Welcome

emocleW

Đảo ngược chuỗi



Đổi cơ số

Khử đệ qui đuôi

……

Lưu giữ đường đi quay lui trong giải thuật tìm đường đi theo chiều sâu

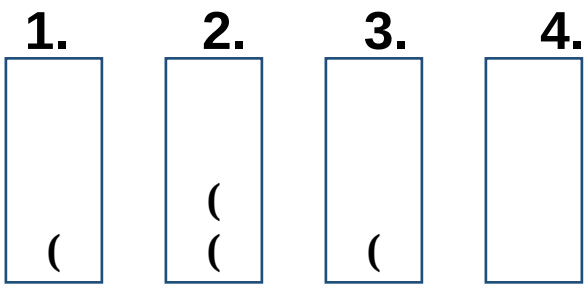
ƯD1: Kiểm tra dấu ngoặc cân bằng

- Kiểm tra dấu ngoặc cân bằng trong biểu thức
  - A + B * (C – D*(B – E) + F): biểu thức có dấu ngoặc cân bằng
  - A + B * (C – D*(B – E)) + F): biểu thức có dấu ngoặc không cân bằng
- Điều kiện để dấu ngoặc cân bằng
  - Khi gặp một dấu “)”, nó phải khớp với một dấu “(” đã gặp trước đó
  - Khi đến cuối chuỗi, mọi dấu “(” đều phải được khớp.

ƯD1: Kiểm tra dấu ngoặc cân bằng

Chuỗi **Ngăn xếp khi giải thuật chạy**

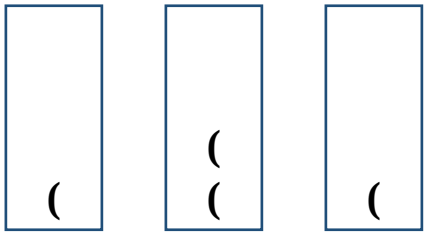
(a*(b+c))



1. push “(”
2. push “(”
3. pop
4. pop

Stack rỗng ⇒ cân bằng

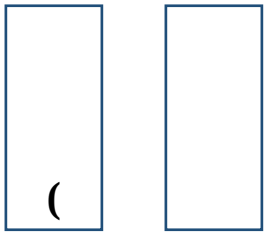
(a*(b+c)



1. push “(”
2. push “(”
3. pop

Stack không rỗng ⇒ không cân bằng

(a*b)+c)



1. push “(”
2. pop

Stack rỗng khi gặp dấu “)” cuối ⇒ không cân bằng

ỨD 2: Tính giá trị biểu thức

- Tính giá trị của biểu thức đại số đơn giản bằng ký pháp Ba Lan đảo (Reverse Polish notation – RPN)
  - Biến đổi biểu thức trung tố sang hậu tố
  - Tính giá trị của biểu thức hậu tố (ký pháp Ba Lan đảo)
- Ví dụ, tính 3 + 4*5 bằng cách chuyển thành 3 4 5 * +

Biểu thức trung tố (infix)

- Toán tử nằm ***giữa*** hai toán hạng
  - Ví dụ, **3 + 4 * 5**
- Nhập nhằng, phải sử dụng dấu ngoặc và độ ưu tiên để xác định thứ tự.
  - Ví dụ, 3 + 4 * 5 = 3 + (4*5) chứ không phải (3 + 4) * 5

Biểu thức hậu tố (postfix)

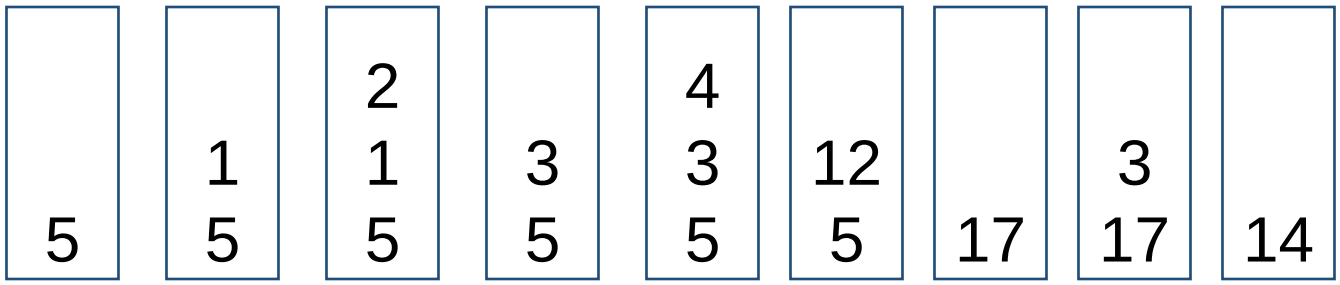
- Toán tử nằm sau hai toán hạng
  - Ví dụ, **3 4 5 + ***
    (4 5 +) // 4 + 5 = 9
    (3 9 *) // 3 * 9 = 27
- Tính giá trị biểu thức ban đầu bằng cách
  - Duyệt biểu thức hậu tố từ trái sang phải
  - Nếu gặp phép toán thì thực hiện phép toán với hai giá trị trước đó và thay hai giá trị này bằng kết quả.
- Biểu diễn hậu tố không nhập nhằng
  - Không cần sử dụng độ ưu tiên cũng như dấu ngoặc
  - Ví dụ, 3 + (4 * 5) $\Rightarrow$ 3 4 5 * + (3 + 4) * 5 $\Rightarrow$ 3 4 + 5 *
  - Các số xuất hiện sau sẽ được tính trước $\Rightarrow$ thích hợp với ngăn xếp

Giải thuật tính biểu thức hậu tố

- Input: chuỗi biểu thức hậu tố q
- Output: một số duy nhất biểu diễn giá trị biểu thức
- Biến cục bộ: ngăn xếp s
- Giải thuật:
  - Duyệt từ đầu đến cuối biểu thức q
  - Với mỗi chuỗi i,
    - Nếu là toán hạng, đẩy vào s
    - Nếu là toán tử, lấy ***hai*** giá trị ra khỏi s, tính kết quả phép toán và đẩy vào s
  - Giá trị duy nhất còn lại ở đỉnh ngăn xếp s sau khi chạy giải thuật là kết quả cuối cùng (nếu khác thì biểu thức không hợp lệ)

Ví dụ tính biểu thức hậu tố

- Ví dụ, tính **5 1 2 + 4 * + 3 −**, tức là 5 + ((1+2)*4) – 3



1. Push 5
2. Push 1
3. Push 2
4. Cộng 1 và 2 (Pop hai giá trị (2, 1) và Push kết quả (3))
5. Push 4
6. Nhân 3 và 4 (Pop hai giá trị (4, 3) và Push kết quả (12))
7. Cộng 5 và 12 (Pop hai giá trị (12, 5) và Push kết quả (17)
8. Push 3
9. Trừ 17 và 3 (Pop hai giá trị (3, 17) và Push kết quả (14))

Gải thuật chuyển trung tố sang hậu tố

- Ví dụ: 5 + ((1+2)*4) – 3 $\Rightarrow$ **5 1 2 + 4 * + 3 –**
- Nhận xét:
  - Nếu xuất hiện dấu ‘(’ thì phải có ‘)’ tương ứng
  - Dấu mở ngoặc xuất hiện sau sẽ được đóng trước
  - Khi gặp toán tử cần xác định toán hạng thứ 2 để đặt phép toán này
- Dùng ngăn xếp để lưu trữ toán tử và dấu mở ngoặc

- Input: biểu thức $\boldsymbol{P}$
- Output: biểu thức kết quả $\boldsymbol{Q}$
- Biến cục bộ: ngăn xếp $\boldsymbol{S}$ lưu phép toán và dấu mở ngoặc ‘(’

Giải thuật chuyển trung tố sang hậu tố

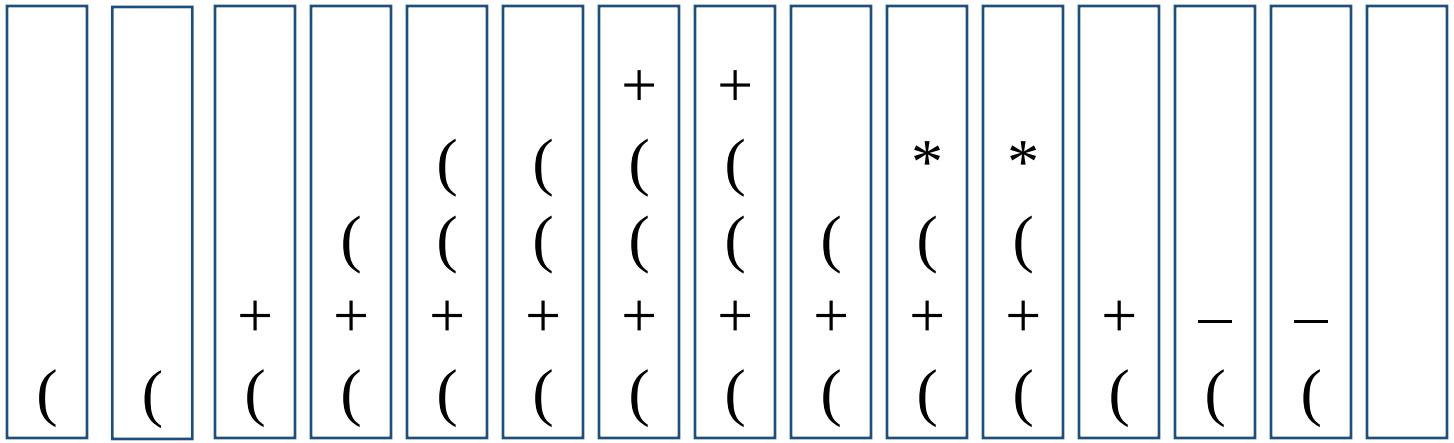
```
push '(' vào S và thêm ')' vào P  // tạo biểu thức mới (P)
while (chưa hết biểu thức P){
  1. đọc 1 kí tự x trong P (từ trái qua phải)
  2. if (x là toán hạng) thêm x vào Q
  3. if (x là dấu ngoặc mở) push x vào S;
  4. if (x là toán tử)
        4.1 while (thứ tự ưu tiên của S.top() ≥ x)
              pop w từ S rồi thêm vào Q
        4.2 push x vào S;
  5. if (x là dấu ngoặc đóng)
        5.1 while (chưa gặp ngoặc mở)
              5.1.1 pop w từ S rồi thêm vào Q
        5.2 pop();              //đẩy ngoặc mở ra khỏi stack
}
```

Ví dụ chuyển trung tố sang hậu tố

Input: 5 + ((1+2) *4) – 3

P (5 + ((1 + 2) * 4) – 3)

S



Q 5 1 2 + 4 * + 3 –

Output: 5 1 2 + 4 * + 3 –

Bài tập rèn luyện

- Push các ký tự A, B, C và D vào ngăn xếp (theo thứ tự được liệt kê) rồi pop chúng ra. Cho biết thứ tự ra khỏi ngăn xếp của các ký tự này.
- Cho hai ngăn xếp, `stack1` và `stack2`, ban đầu đều rỗng. Tình trạng của các ngăn xếp này sẽ như thế nào sau khi thực hiện các thao tác dưới đây?

```
1. stack1.push(1)
2. stack1.push(2)
3. stack2.push(3)
4. stack2.push(4)
5. stack1.pop()
6. stackTop = stack2.peek()
7. stack1.push(stackTop)
8. stack1.push(5)
9. stack2.pop()
10. stack2.push(6)
```

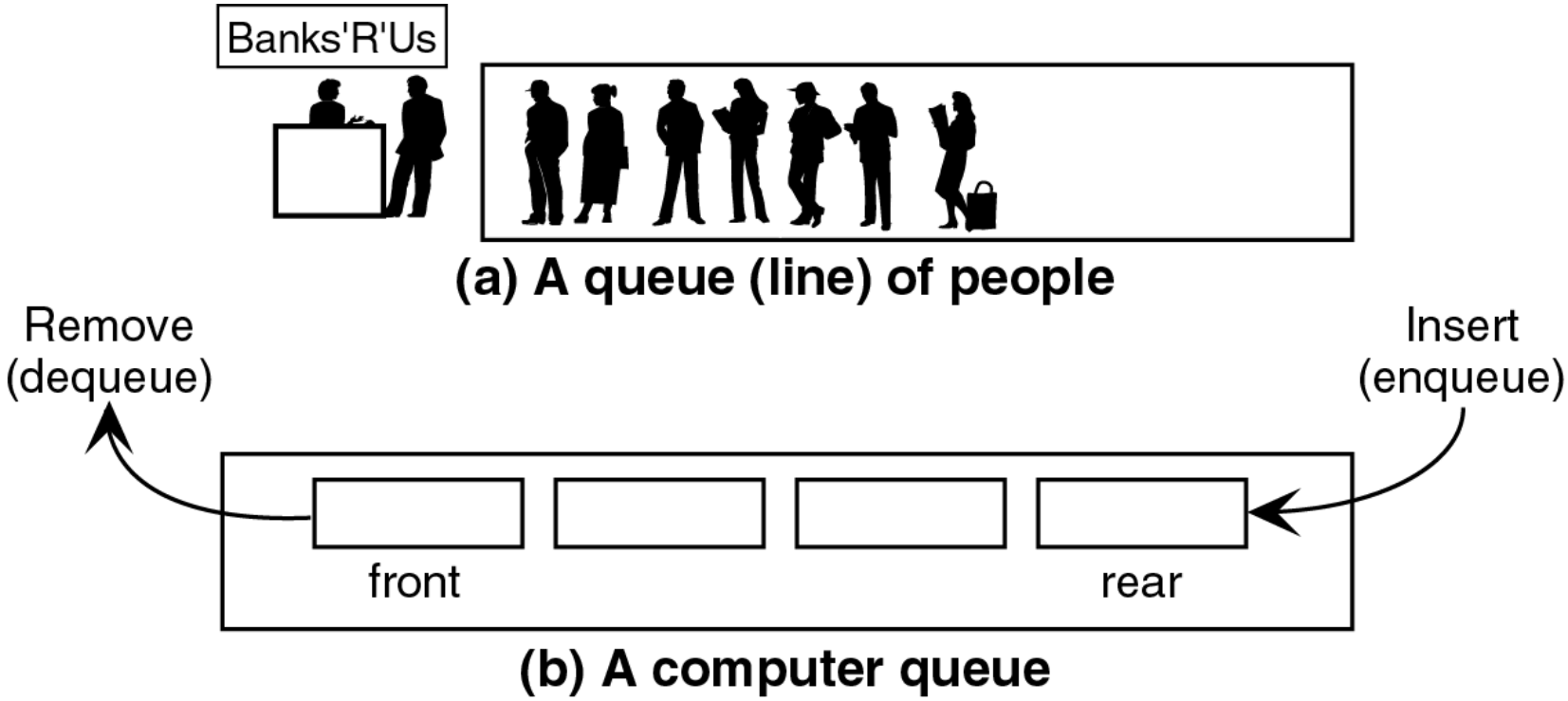
Bài tập rèn luyện

- Với mỗi chuỗi bên dưới, trình bày các bước kiểm tra dấu ngoặc cân bằng và thể hiện nội dung ngăn xếp tương ứng với mỗi bước.
  - `x{{yz}}`
  - `{x{y{{z}}}`
  - `{{{x}}}`
- Trình bày các bước thực hiện và nội dung ngăn xếp tương ứng khi tính giá trị biểu thức bằng phương pháp RPN.

  (5*2 +1) – (4/2+7)
- Cài đặt ngăn xếp bằng mảng và danh sách liên kết

Hàng đợi (Queue)

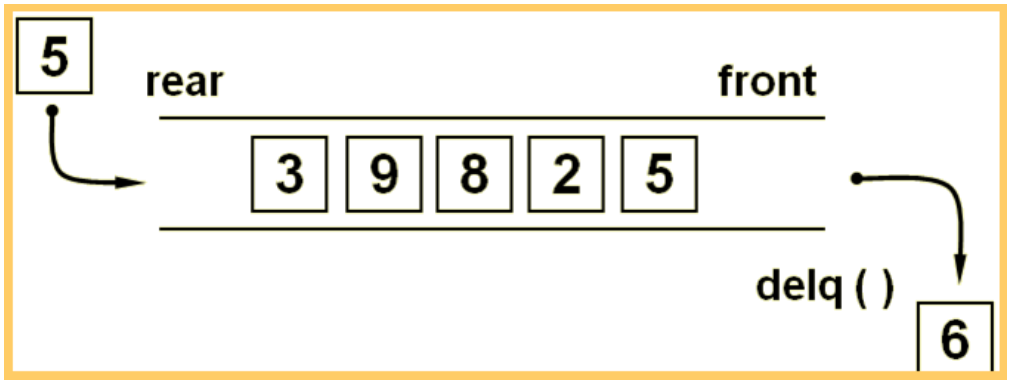
Hàng đợi



- Hàng đợi được triển khai phổ biến trong đời thực.
- Ta chỉ có thể thao tác tại một đầu của hàng đợi.

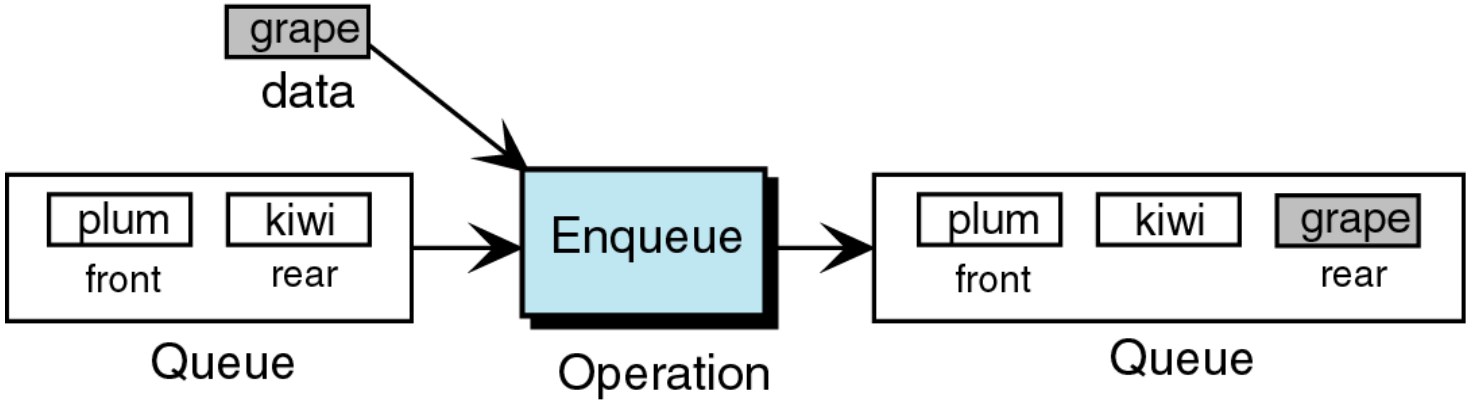
Định nghĩa hàng đợi

- **Hàng đợi** là cấu trúc dữ liệu trừu tượng (ADT), được sử dụng trong nhiều ngôn ngữ lập trình.
  - Ví dụ, C++ std::queue, C# Collections.Queue, Java java.util.* Queue
- Hàng đợi gồm nhiều phần tử có thứ tự theo cơ chế FIFO.
  - **First In First Out:** phần tử được đặt vào hàng đợi sớm nhất sẽ được lấy ra đầu tiên.
  - Phần tử mới đi vào ở phía sau (rear) của hàng đợi, các phần tử đi ra ở phía trước (front) của hàng đợi.



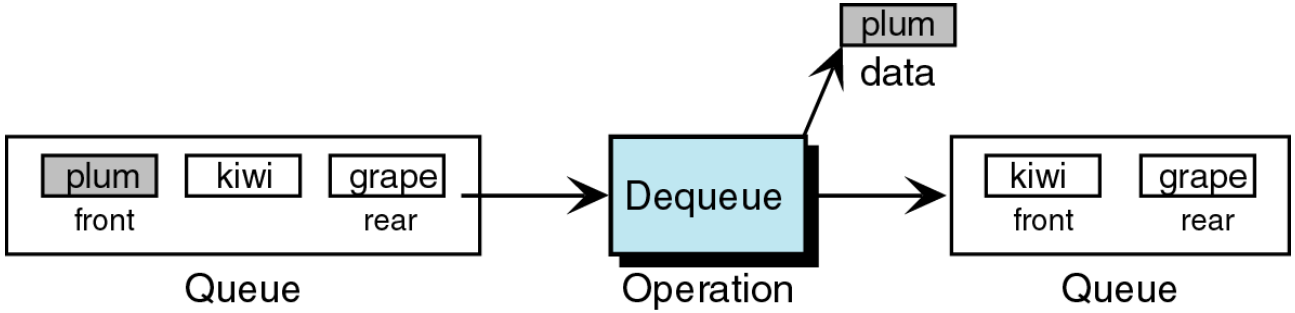
Thao tác cơ bản trên hàng đợi

- **`isEmpty()`:** kiểm tra tình trạng của hàng đợi
- **`enqueue(newEntry)`:** thêm một phần tử `newEntry` vào (cuối) hàng đợi, có thể làm hàng đợi đầy

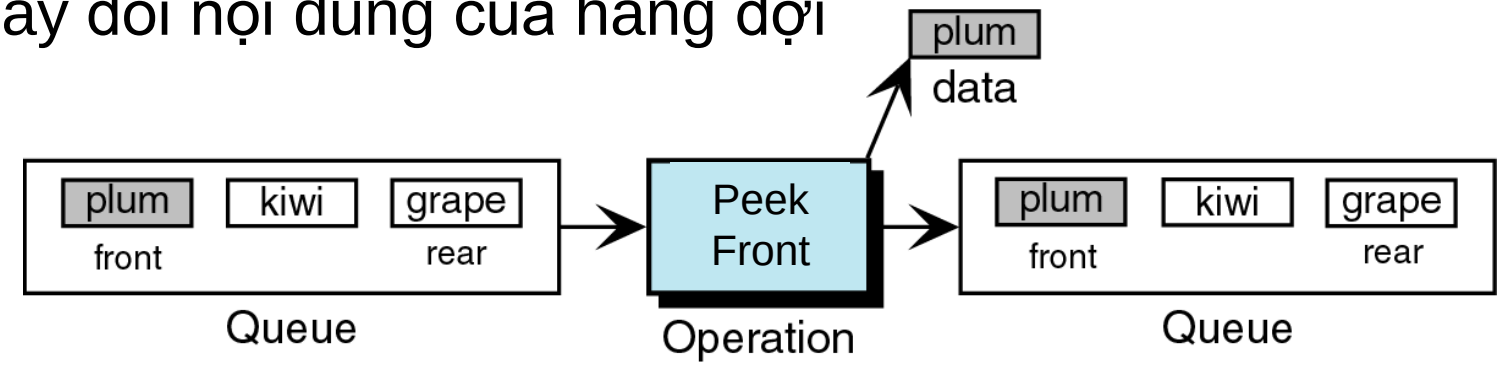


Thao tác cơ bản trên hàng đợi

- **`dequeue()`:** lấy một phần tử từ (đầu) hàng đợi, có thể làm hàng đợi rỗng



- **`peekFront()`:** xem phần tử ở đầu hàng đợi nhưng không làm thay đổi nội dung của hàng đợi



Ví dụ áp dụng hàng đợi

- Cho chuỗi thao tác sau:

EAS*Y**QUE***ST***I*ON

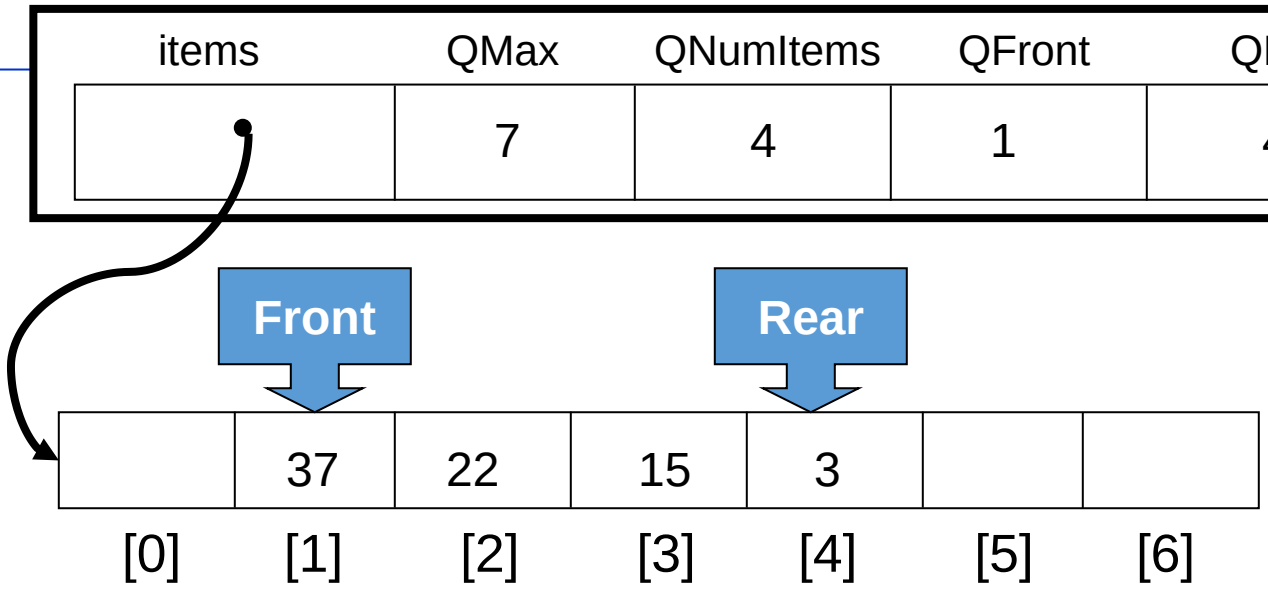
trong đó

  - *ký tự*: đẩy ký tự tương ứng vào hàng đợi
  - *: lấy một phần tử trong hàng đợi và xuất ra màn hình

- Cho biết kết quả cuối cùng trên màn hình?

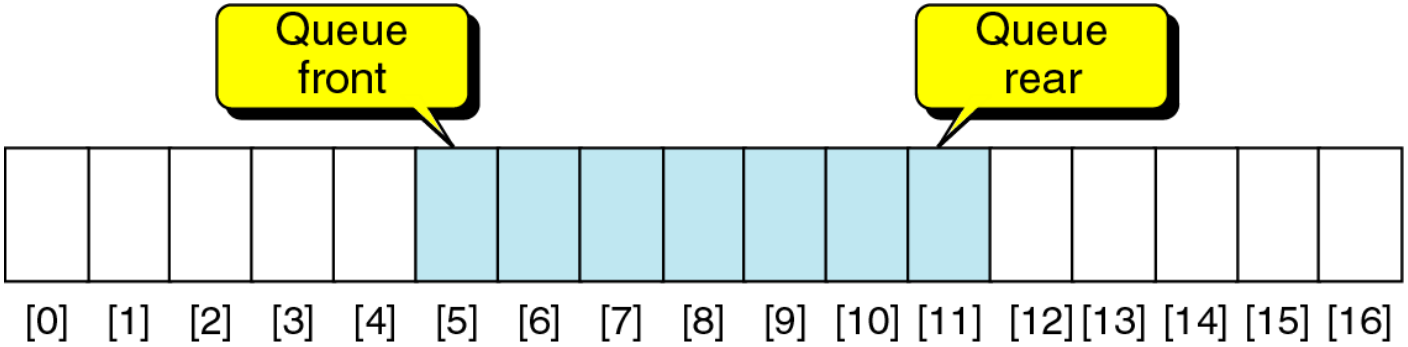
Cài đặt hàng đợi bằng mảng

```
template<class ItemType>
class ArrayStack{
    ItemType items[MAX_ QUEUE]; // Array of queue items
    int front;          // Index to front of queue
    int back;           // Index to back of queue
    int count;      // Number of items currently in the queue
}
```

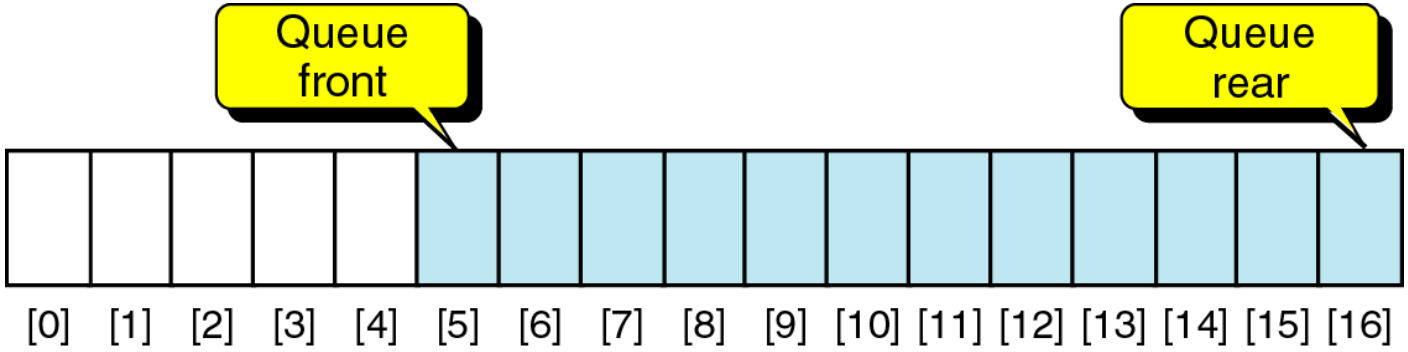


Cài đặt hàng đợi bằng mảng

- Các phần tử đang chứa trong hàng đợi



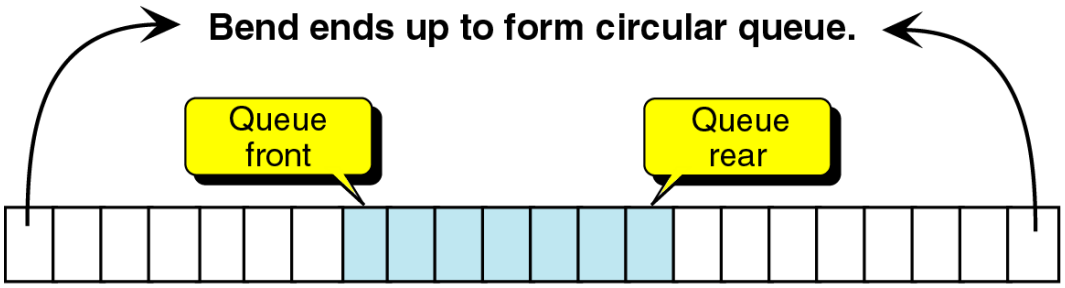
- Khi thêm nhiều phần tử sẽ làm "tràn" mảng → "Tràn giả"

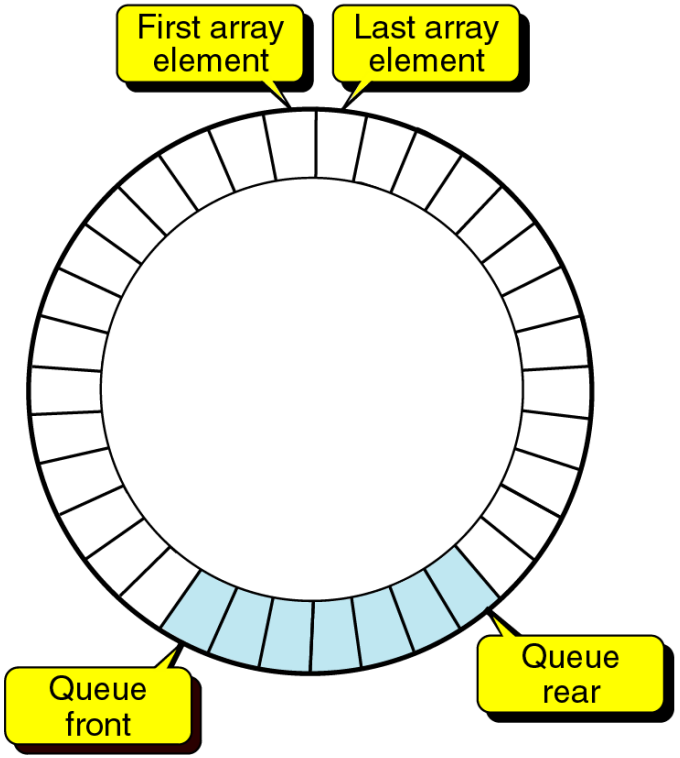


- Giải pháp là gì ? Có nên Sử dụng 1 mảng vô cùng lớn ?

Cài đặt hàng đợi bằng mảng

- Xử lý mảng như là 1 danh sách vòng





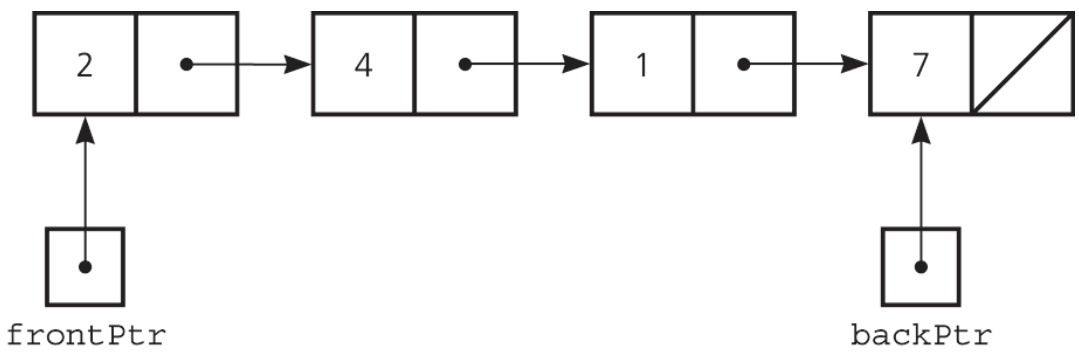
Cài đặt hàng đợi bằng mảng

```
template<class ItemType>
bool ArrayQueue<ItemType>
::enqueue(const ItemType& newEntry)
{
      bool result = false;
      if (count < MAX_QUEUE)
      {
            // Queue has room for another item
            back = (back + 1) %
                        MAX_QUEUE;
            items[back] = newEntry;
            count++;
            result = true;
      } // end if
      return result;
} // end enqueue
```

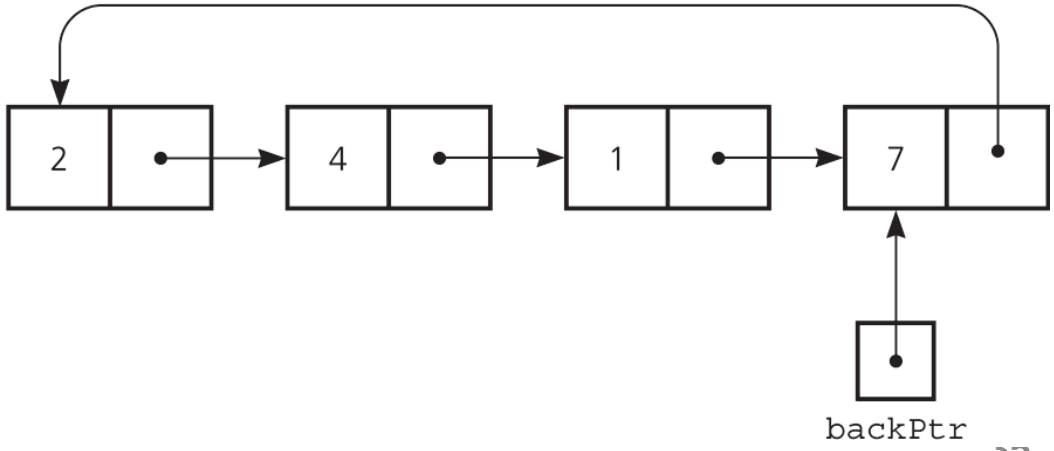
```
template<class ItemType>
bool ArrayQueue<ItemType>
:: dequeue()
{
      bool result = false;
      if (!isEmpty())
      {

            front = (front + 1) %
                        MAX_QUEUE;

            count--;
            result = true;
      } // end if
      return result;
} // end dequeue
```

Cài đặt hàng đợi bằng DSLK

- Danh sách liên kết đơn với hai con trỏ ngoài
  - Một trỏ đến phía trước danh sách, một trỏ đến phía sau danh sách



- Danh sách vòng với một con trỏ ngoài
  - Một trỏ đến phía sau danh sách

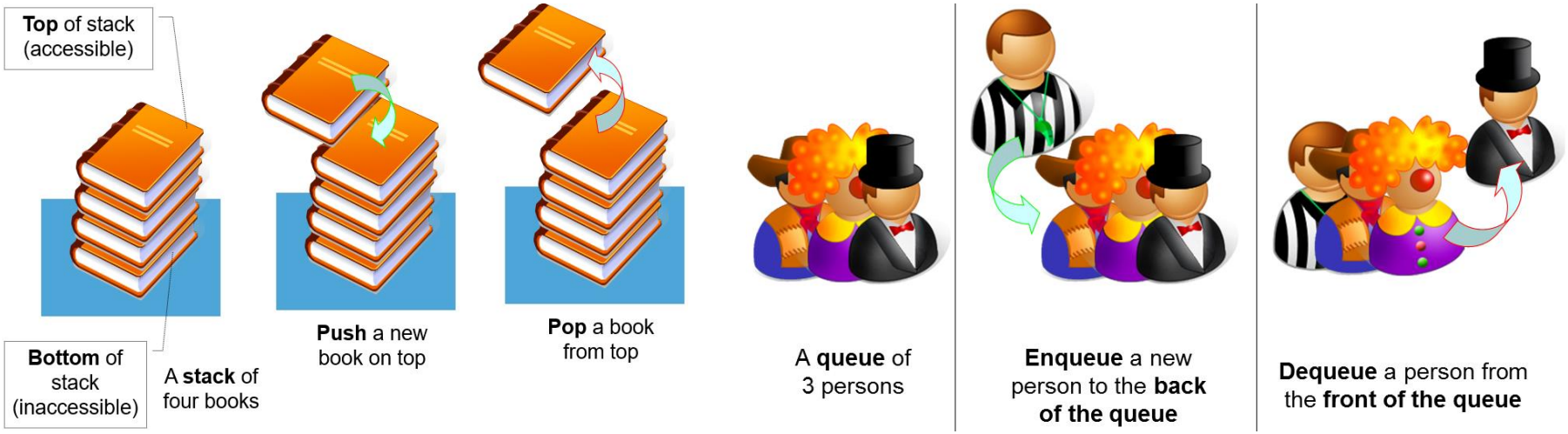


Ứng dụng của hàng đợi

- Quản lý việc thực hiện các tác vụ trong môi trường xử lý song song
- Hàng đợi in ấn các tài liệu
- Vùng nhớ đệm (buffer) dùng cho bàn phím
- Quản lý thang máy

Ngăn xếp và hàng đợi

- Ngăn xếp và hàng đợi rất tương tự nhưng đối lập nhau

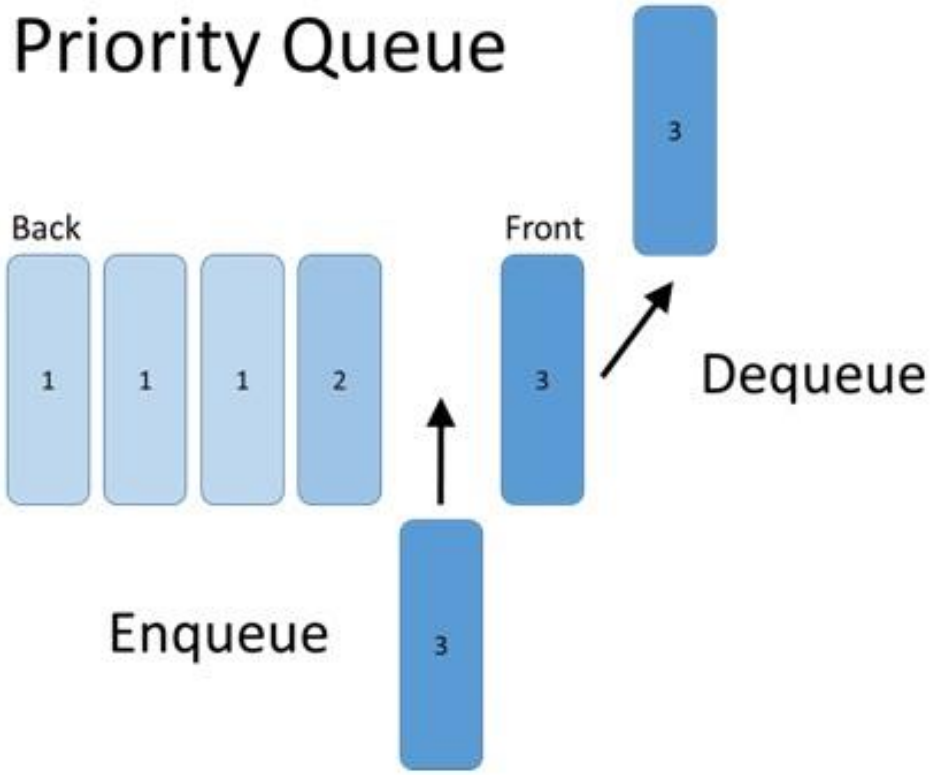


- Các thao tác của ngăn xếp và hàng đợi đi thành từng cặp

Ngăn xếp	Hàng đợi
push	enqueue
pop	dequeue
peek	peekFront

Hàng đợi ưu tiên

- **Hàng đợi ưu tiên** là hàng đợi trong đó mỗi phần tử có một độ ưu tiên xác định và phần tử lấy ra khỏi hàng đợi luôn có độ ưu tiên cao nhất.

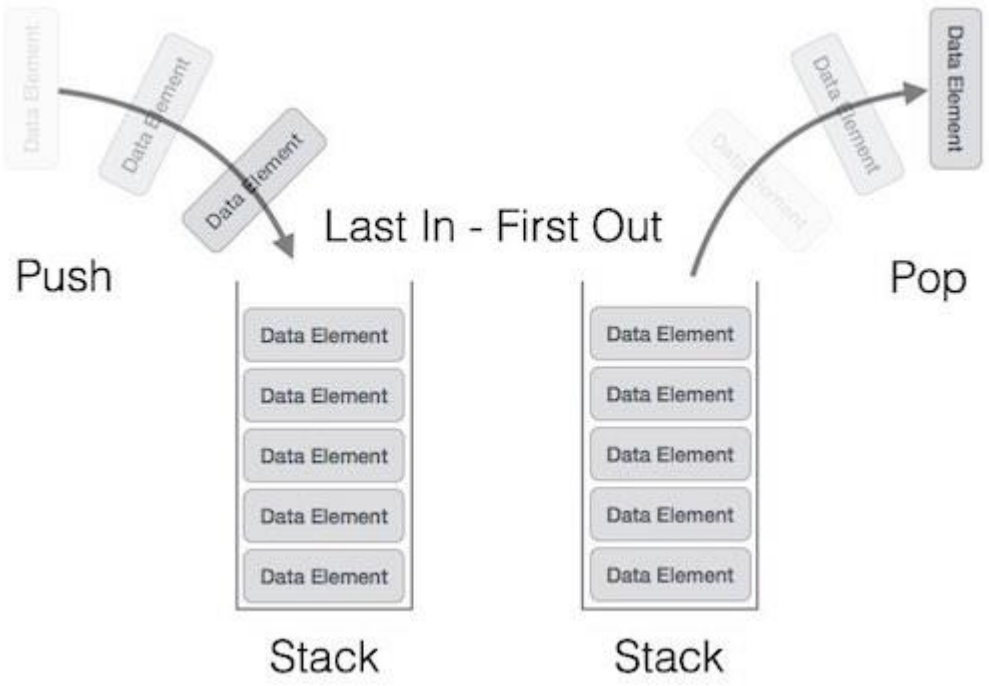


Bài tập rèn luyện

- Enqueue các ký tự A, B, C và D vào hàng đợi (theo thứ tự được liệt kê) rồi dequeue chúng. Cho biết thứ tự ra khỏi hàng đợi của các ký tự này.
- Cho hai hàng đợi, `queue1` và `queue2`, ban đầu đều rỗng. Tình trạng của các hàng đợi này sẽ như thế nào sau khi thực hiện các thao tác dưới đây?

```
1. queue1.enqueue(1)
2. queue1.enqueue(2)
3. queue2.enqueue(3)
4. queue2.enqueue(4)
5. queue1.dequeue()
6. queueFront = queue2.peekFront()
7. queue1.enqueue(queueFront)
8. queue1.enqueue(5)
9. queue2.dequeue()
10. queue2.enqueue(6)
```

- Cài đặt hàng đợi bằng mảng và danh sách liên kết



Bảng băm (Hash table)

Bài toán tổng quát

- Cho một tập S các phần tử được đặc trưng bởi giá trị khóa
  - Trên các giá trị khóa đó xác định một quan hệ thứ tự
- Yêu cầu
  - Tổ chức S như thế nào để việc tìm kiếm một phần tử có khóa $\boldsymbol{k}$ cho trước tốn ít công sức nhất trong giới hạn bộ nhớ cho phép.

Bài toán tổng quát

Keys

John Smith

Lisa Smith

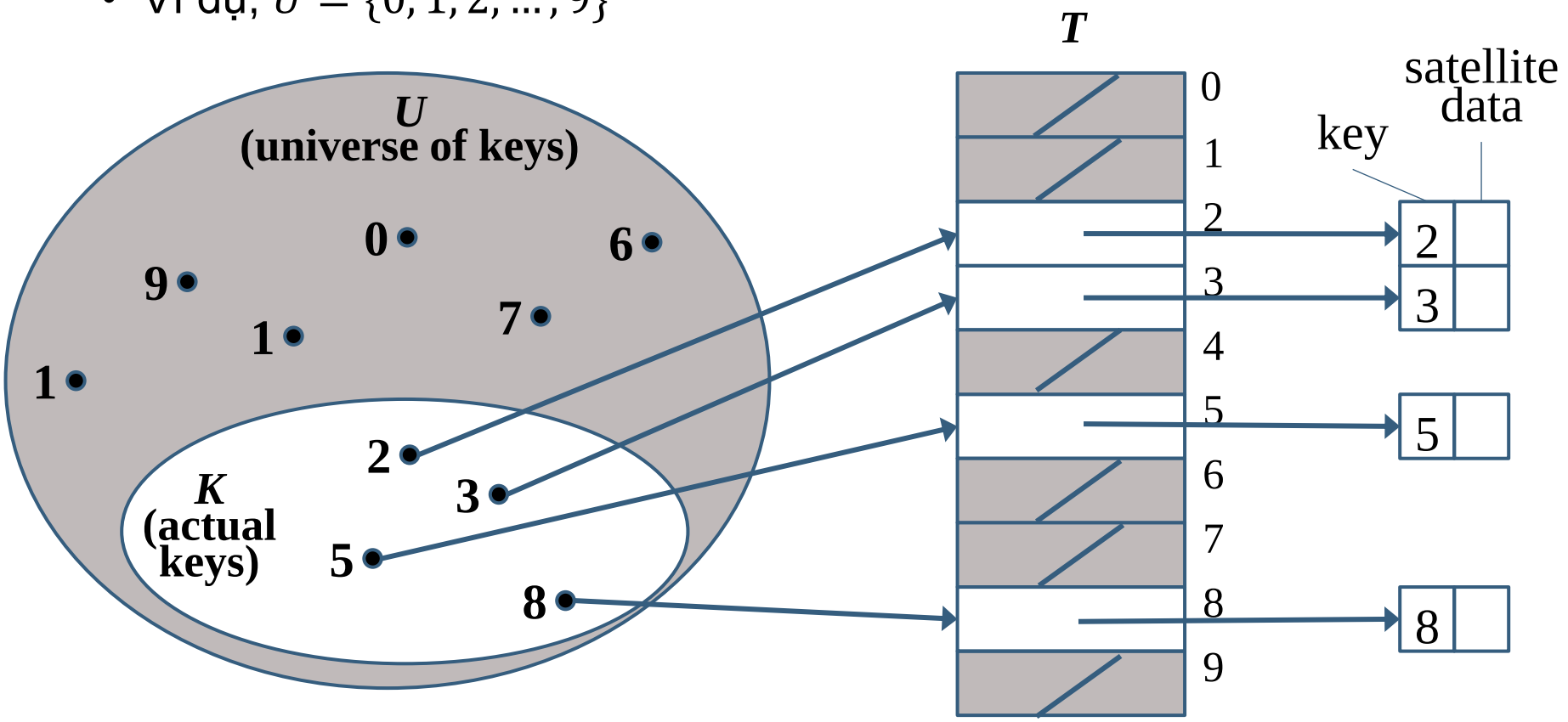
Sam Doe

Key-value pairs
(records)

Lisa Smith	+1-555-8976
John Smith	+1-555-1234
Sam Doe	+1-555-5030

Direct-address table

- Giả sử có một tập khoá U
  - Kích thước không quá lớn, các giá trị khoá phân biệt
  - Ví dụ, $U = \{0, 1, 2, \ldots, 9\}$



Mô hình minh họa dùng direct-address table $T[m]$ để lưu trữ các khoá trong tập U

Direct-address table

- Mô tả ý tưởng
  - Mảng $T[m]$ ($T[0], \dots, T[m-1]$) chứa khoá trong U sao cho $|T| = |U|$
  - Mỗi vị trí $T[k]$ (slot) sẽ chứa
    - Khóa k, hay
    - NULL nếu khoá k không có trong tập hợp
- Lưu ý:
  - U (Universe of keys): tập các giá trị khóa
  - K (Actual keys) $\subseteq U$: tập các khoá thực sự được dùng
- Chi phí thao tác: $\boldsymbol{O(1)}$

Giới hạn của direct-address table

- Kích thước tập U quá lớn $\rightarrow$ không thể tạo bảng T với số slot tương ứng với $|U|$.
- Kích thước của tập K quá nhỏ so với U $\rightarrow$ rất nhiều slot bị bỏ trống.

Một ý tưởng khác

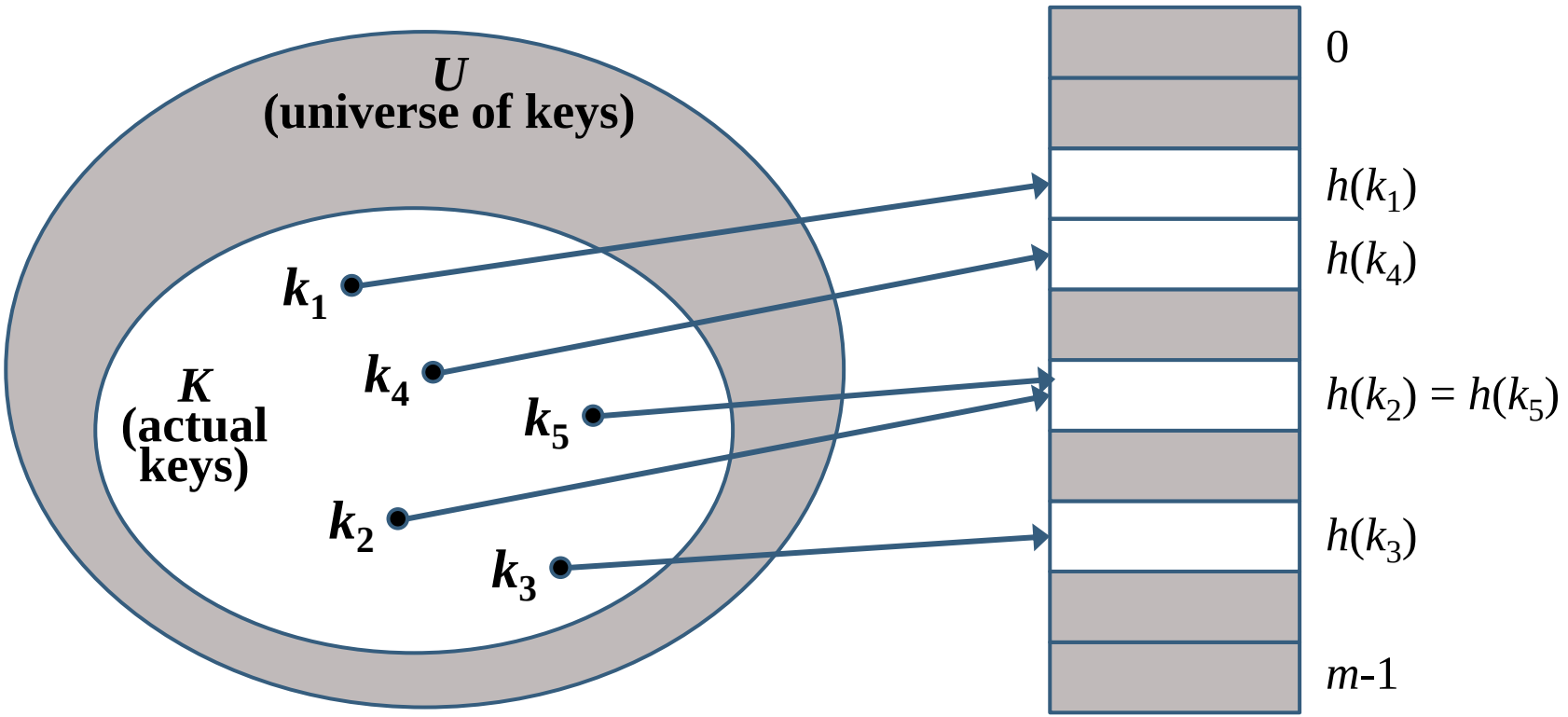
- Khi tập khóa K nhỏ hơn nhiều so với tập U $\rightarrow$ chỉ dùng mảng $T[m]$ với kích thước vừa đủ cho tập K, $m = \Theta(|K|)$.
- Do đó, không thể áp dụng ánh xạ trực tiếp $T[k] \leftarrow k$ được.
- Thay vì ánh xạ trực tiếp $T[k] \leftarrow k$, ta dùng hàm h để ánh xạ $T[h(k)] \leftarrow k$.

Hàm băm (hash function)

- **Hàm băm h** (hash function) ánh xạ các khoá của U vào những slot của bảng băm $T[0..m-1]$.

$$h: U \to \{0, 1, \ldots, m-1\}$$

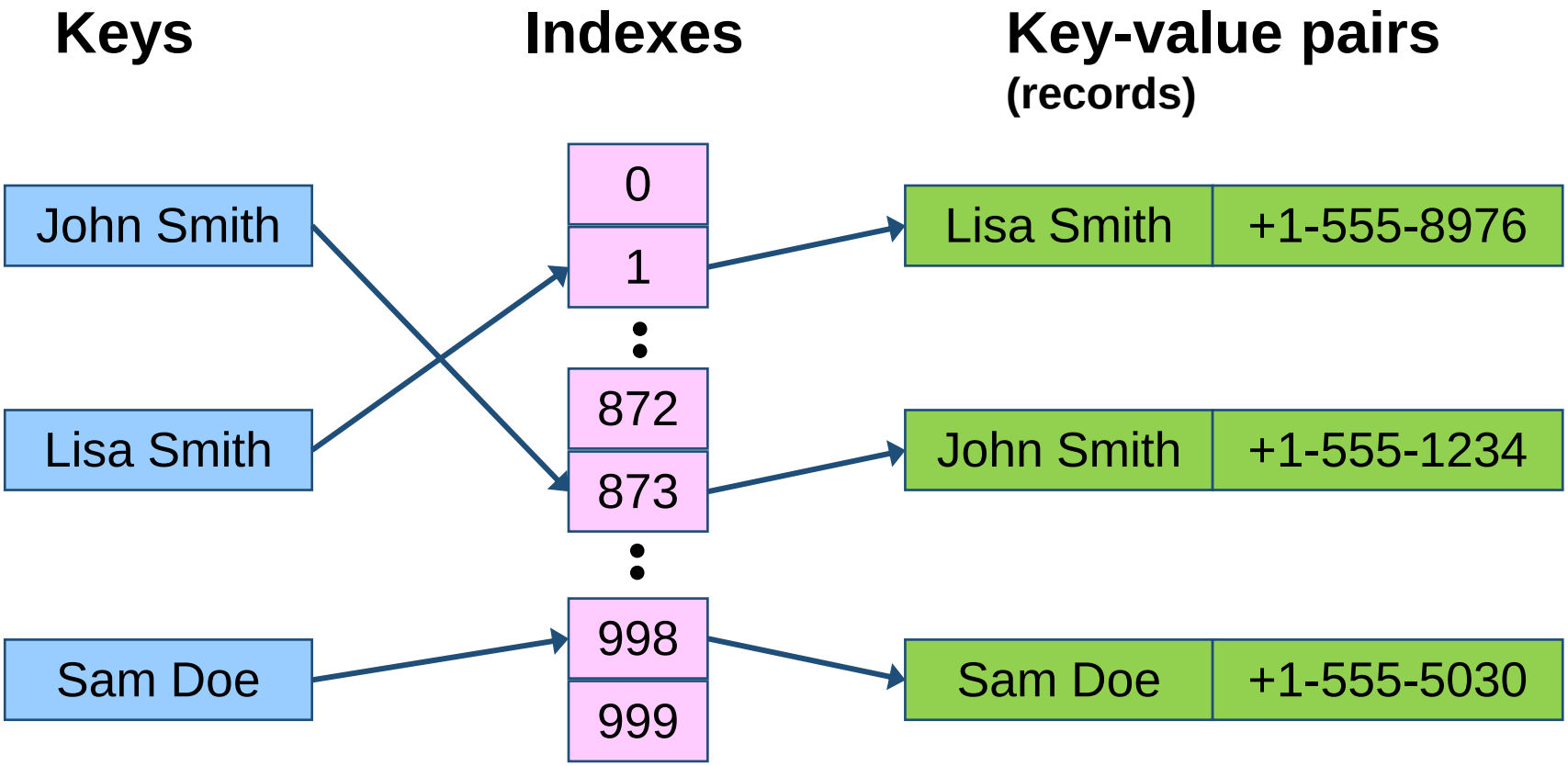
  - $h(k)$: giá trị băm (hash value) của khoá k



Bảng băm (hash table)

- **Bảng băm** (hash table) là một cấu trúc dữ liệu, lưu trữ các khóa trong bảng T (danh sách đặc); sử dụng một hàm băm để ánh xạ khóa với một địa chỉ lưu trữ.
  - Hàm băm có tác dụng biến đổi khoá thành chỉ số địa chỉ (index) – tương ứng với khoá
- Cấu trúc bảng băm rất phù hợp để cài đặt bài toán "từ điển"
  - Bài toán "từ điển" (Dictionary): dạng bài toán chỉ chủ yếu sử dụng thao tác chèn thêm (Insert) và tìm kiếm (Search).

Ví dụ về bảng băm



Tính chất cơ bản của bảng băm

- Cấu trúc lưu trữ dùng trong bảng băm thường là danh sách đặc – mảng hay tập tin.
- Thao tác cơ bản nhất được cung cấp bởi bảng băm là **Tìm kiếm** (lookup).
- Chi phí tìm kiếm trung bình là $O(1)$
  - Bất kể số lượng phần tử của mảng
- Chi phí tìm kiếm xấu nhất (ít gặp) có thể là $O(n)$.

Khai báo cấu trúc HashTable

```
template <class T> class HASH_TABLE {
private:
      T *items; // array of hash items
      int maxSize; // maximum size of hash table
      unsigned long hash(T key); // hash function
public:
      HASH_TABLE(int m); // create hash table with m slots
      HASH_TABLE(const HASH_TABLE &aHashTable);
      ~HASH_TABLE(); // destructor
      // operations
      bool insert(T newItem);
      bool remove(T key);
      bool retrieve(T key, T &item);
}; // end class
```

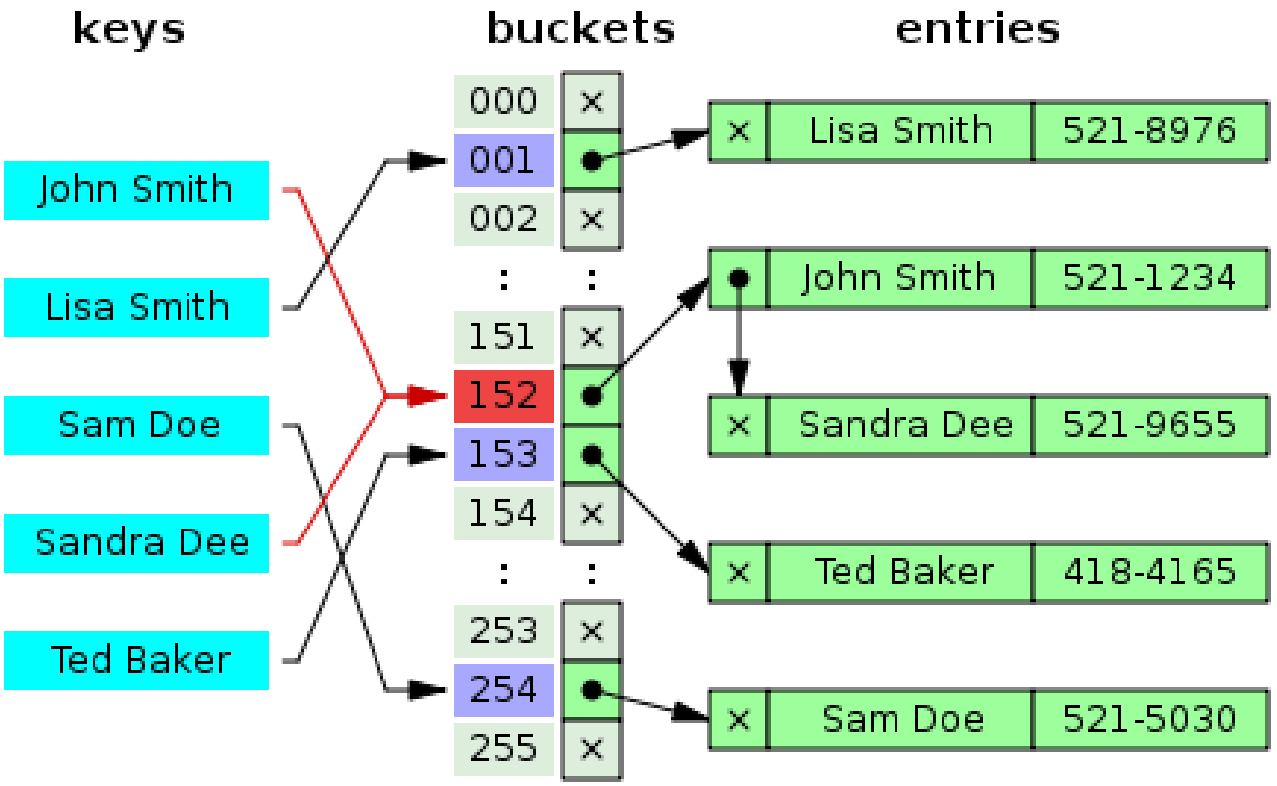
Xung đột địa chỉ trong bảng băm

- Một cách lý tưởng, hàm băm sẽ ánh xạ mỗi khoá vào một slot riêng biệt của bảng T
- Tuy nhiên, điều này trong thực tế khó đạt được
  - Tập giá trị khóa (U) có thể lớn hơn số khóa thực tế sử dụng (K) rất nhiều, $m \ll |U|$.
  - Các khoá là không biết trước
  - Ví dụ, danh sách sinh viên gồm 3000 SV, MSSV gồm 7 chữ số

$$\rightarrow U = 10^7 \text{ khóa so với } K = 3000 \text{ khóa hữu dụng}$$

Xung đột địa chỉ trong bảng băm

- Hầu hết cấu trúc bảng băm trong thực tế đều chấp nhận một tỉ lệ nhỏ các khoá đụng độ (collision) và xây dựng phương án giải quyết sự đụng độ đó



Minh họa sự đụng độ và phương án giải quyết "chaining (móc xích)

Xây dựng hàm băm

- Đây là thành phần quan trọng nhất của bảng băm.
- Hàm băm biến đổi khóa k của phần tử thành địa chỉ trong bảng băm.
  - Ví dụ, $h(k) = k \bmod m$
- Khóa có thể là dạng số hay dạng chuỗi
- Phương án xử lý chính của hàm băm là xem các khóa như là các số nguyên
  - Khóa là chuỗi “key” $\rightarrow$ xử lý 3 thành phần 107 (k), 101 (e), và 121 (y)

Xây dựng hàm băm

- Hàm băm là hàm nhiều – một (many-to-one function).
- Một hàm băm tốt là yếu tố tiên quyết để tạo ra bảng băm hiệu quả.
- Các yêu cầu cơ bản đối với hàm băm:
  - Tính toán nhanh, dễ dàng
  - Các khóa được phân bố đều trong bảng
  - Ít xảy ra đụng độ
- Ví dụ, danh sách có $m = 1000$ sinh viên, $h(k) = k \bmod m$
  - Hàm băm thỏa mãn yêu cầu tính toán nhanh và trải đều trên bảng.

Xây dựng hàm băm bằng phép mod

- Xây dựng hàm băm bằng phép chia modular

$$\boldsymbol{h(k) = k \bmod m}$$

- Chọn m như thế nào ?
  - m không được là lũy thừa của 2. Nếu $m = 2^p$ thì $h(k) = k \bmod m$ chính là p bit thấp của k.
  - m không nên là lũy thừa của 10, vì khi đó, hash value sẽ không sử dụng tất cả chữ số thành phần của k
  - Nên chọn m là số nguyên tố nhưng không quá gần với giá trị 2^n
- Ví dụ, $h(k) = k \bmod 11$

Xây dựng hàm băm bằng phép nhân

- Xây dựng hàm băm bằng phép nhân

$$\boldsymbol{h(k) = \lfloor m * (k * A \bmod 1) \rfloor}$$

  - trong đó, $0 < A < 1$, $(k * A \bmod 1)$ là phần thập phân của $(k * A)$, $\lfloor x \rfloor$ là floor(x).
- Chọn m như thế nào ?
  - Giá trị m không quan trọng, ta thường chọn $m = 2^p$
- Knuth đã phân tích và đưa ra giá trị A tối ưu

$$A \approx \frac{(\sqrt{5} - 1)}{2} = 0.6180339887$$

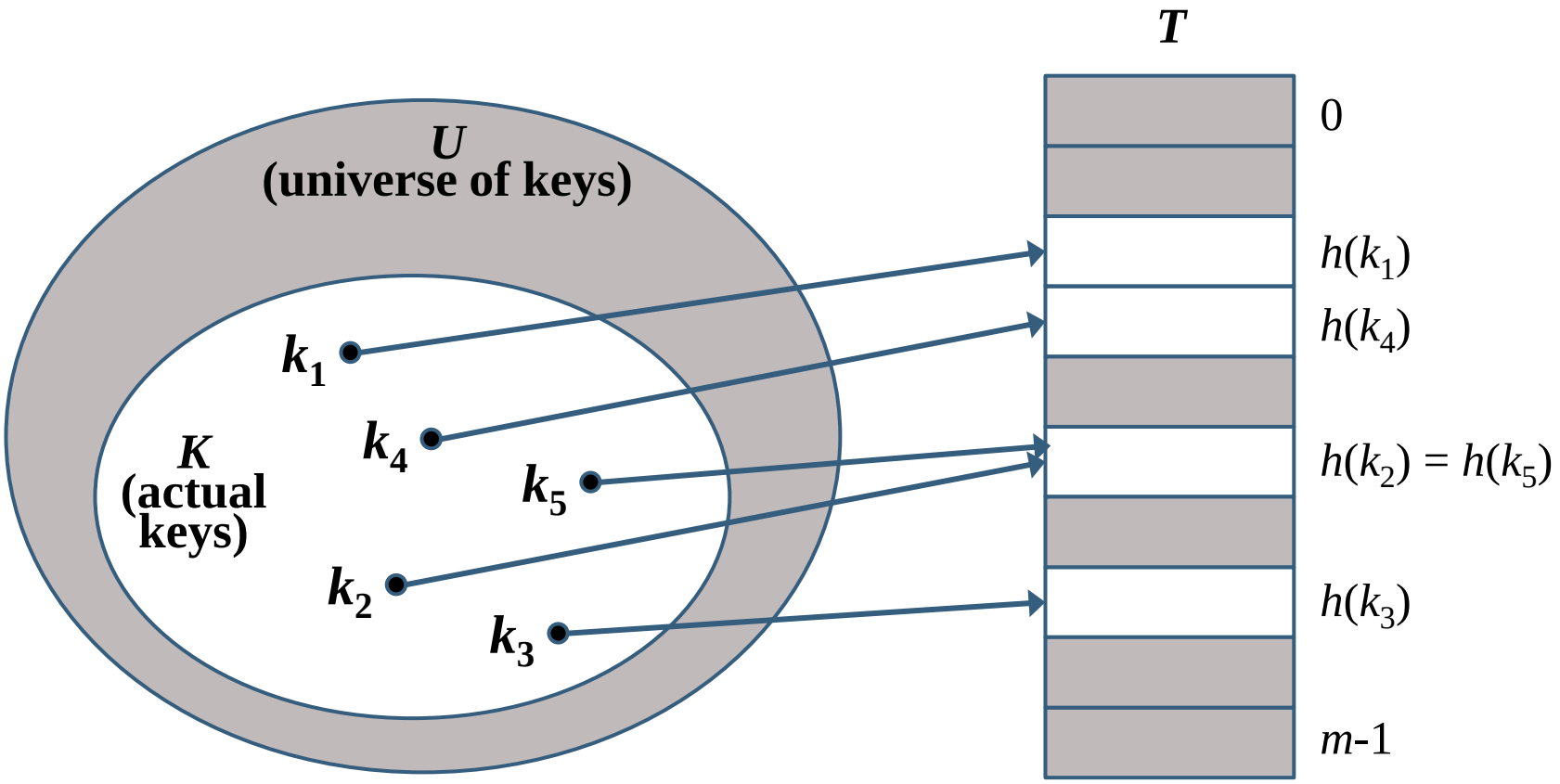
Xây dựng hàm băm bằng phép nhân

- Ví dụ, giả sử $k = 123456, m = 10000$, A như trên

$$h(k) = \lfloor 10000 * (123456 * 0.6180339887 \bmod 1 \rfloor$$
$$= \lfloor 10000 * (76300.0041151 \bmod 1 \rfloor$$
$$= \lfloor 10000 * 0.0041151 \rfloor$$
$$= \lfloor 41.151 \rfloor$$
$$= 41$$

Sự đụng độ trong bảng băm

- Sự đụng độ là tình huống $\exists k_1, k_2 \in K: k_1 \neq k_2, h(k_1) = h(k_2)$



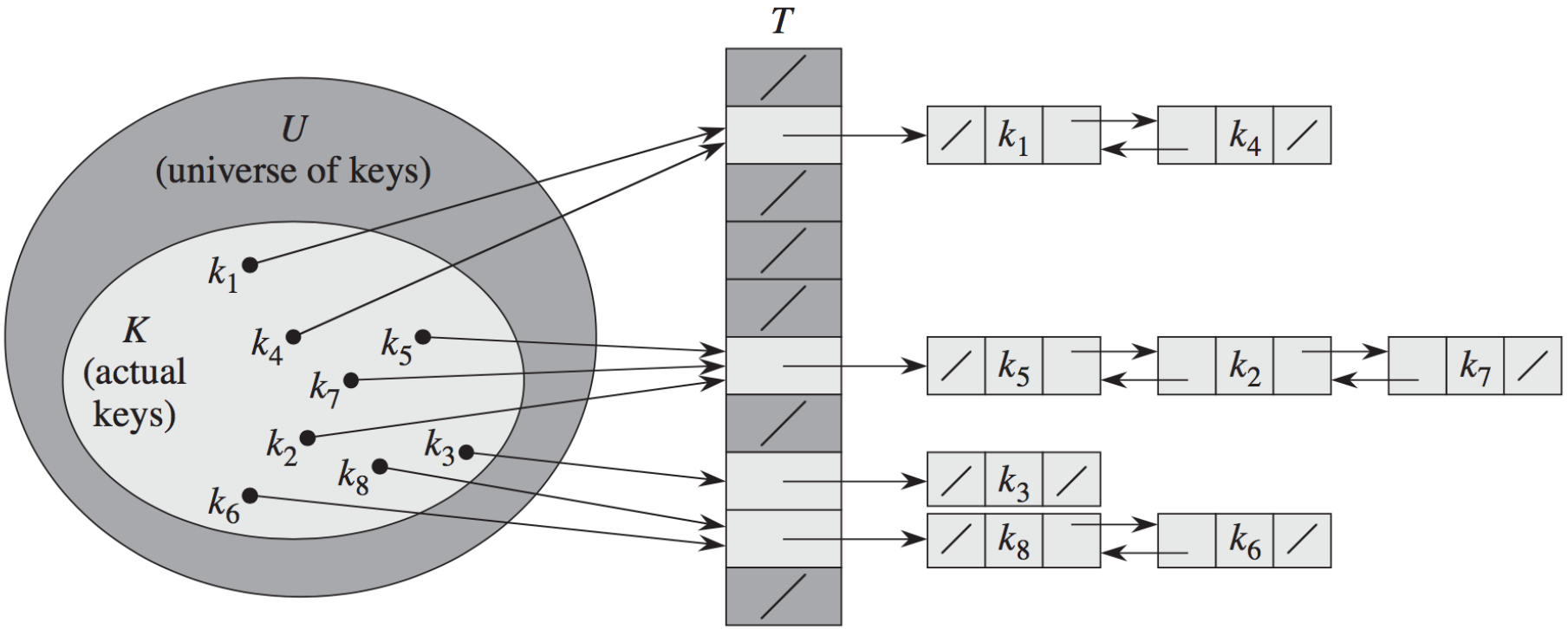
Đụng độ giữa khóa k_2 và k_5

Các phương pháp xử lý đụng độ

- Phương pháp nối kết (Chaining)
- Phương pháp địa chỉ mở (Open-addressing)

Phương pháp nối kết

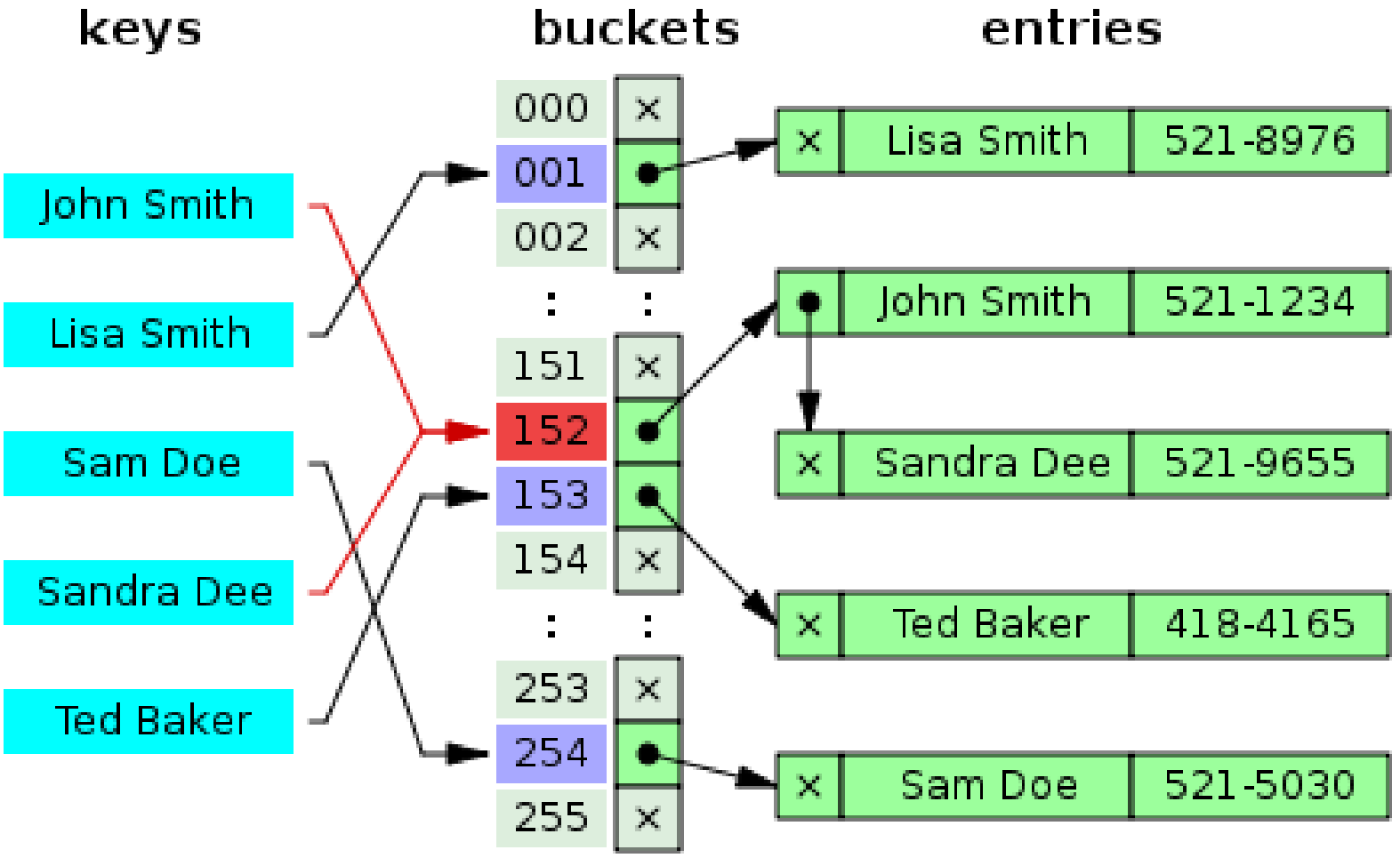
- Đưa tất cả các khóa đụng độ vào một slot, lưu thành một danh sách liên kết.



Phương pháp nối kết

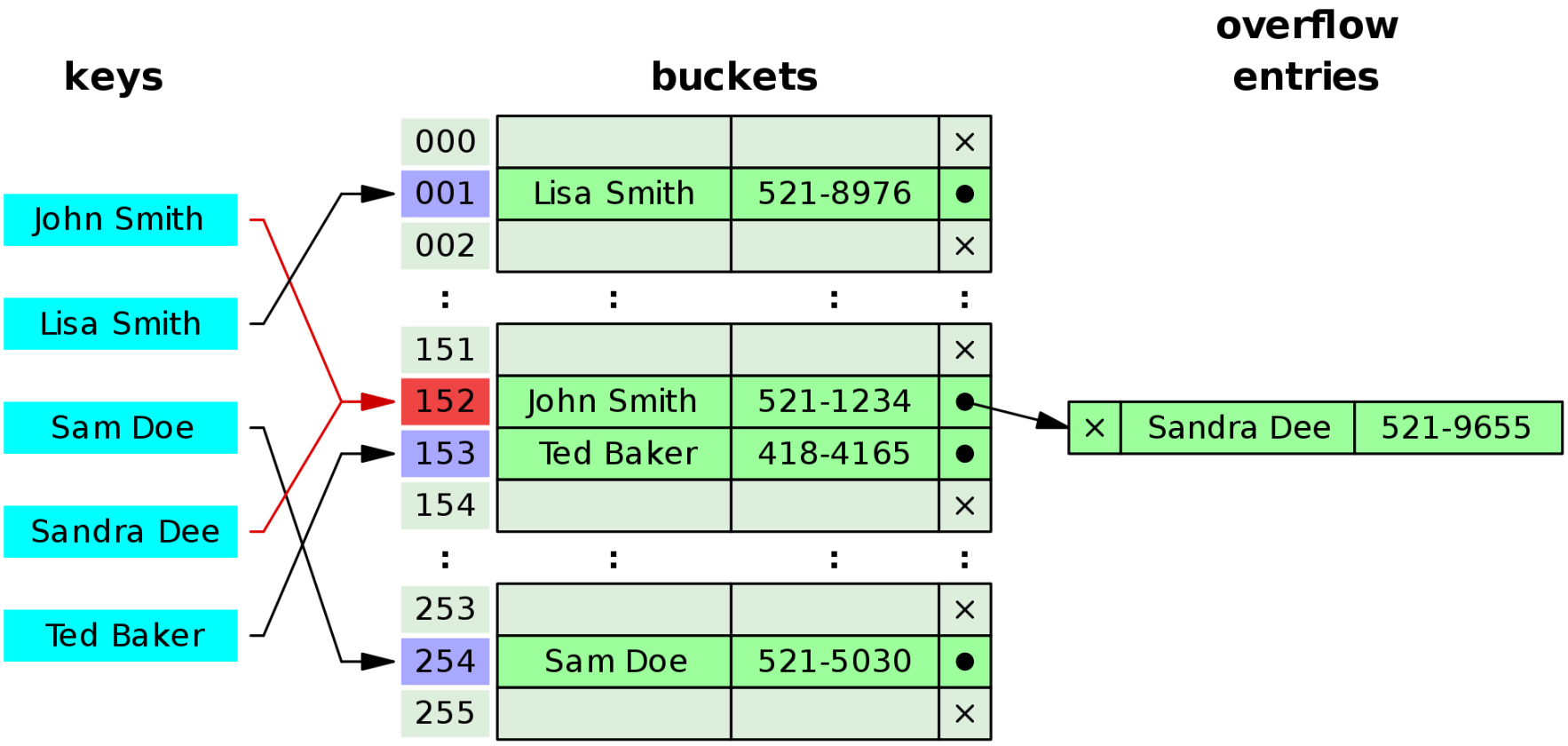
- Ứng với mỗi địa chỉ của bảng, ta có một danh sách liên kết chứa các phần tử có khóa khác nhau nhưng có cùng địa chỉ
- Như vậy, bảng băm là một danh sách gồm m phần tử chứa địa chỉ đầu của các danh sách liên kết.

Ví dụ phương pháp nối kết



Bảng T chỉ lưu con trỏ của danh sách liên kết

Ví dụ phương pháp nối kết



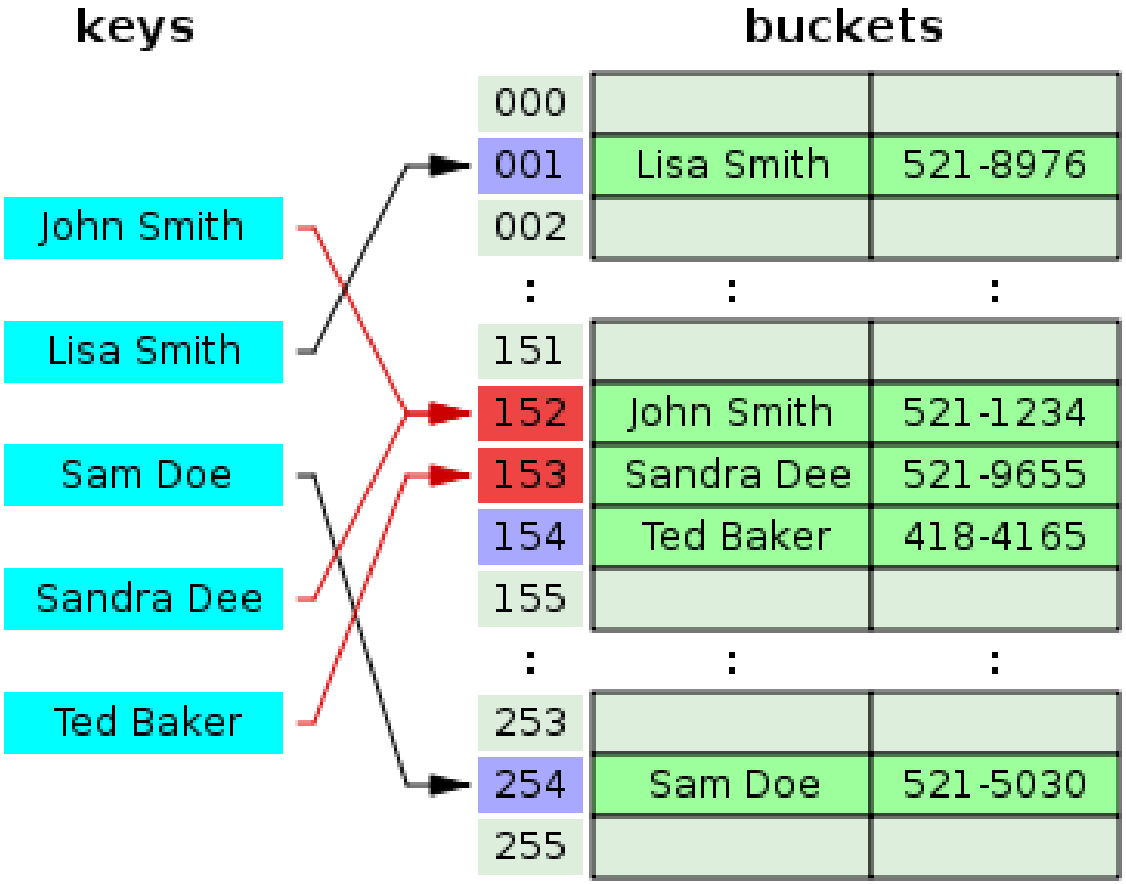
Bảng T lưu phần tử đầu tiên + con trỏ của danh sách liên kết

Phương pháp nối kết

- Chi phí các thao tác
  - Insert: chi phí xấu nhất là $O(1)$
  - Search và Delete: chi phí trung bình là $\Theta(1 + \alpha)$
    - $\alpha = n/m$ (load factor: số phần tử trung bình lưu trữ trong một slot)
- Cấu trúc dữ liệu khác
  - Ngoài danh sách liên kết, ta có thể áp dụng các cấu trúc khác hiệu quả hơn (khi tìm kiếm) như: cây cân bằng (AVL, Red-Black, AA), hay mảng cấp phát động,…

Phương pháp địa chỉ mở

- Khi đụng độ xảy ra, tìm đến vị trí kế tiếp nào đó trong bảng cho đến khi tìm thấy vị trí nào còn trống.



Thuật toán cơ bản trên bảng băm

Thuật toán cơ bản để thêm khóa k

```
Hash-Insert(T, k)
1. i ← 0
2. repeat j ← h(k,i)
3.     if T[j] = NIL
4.     then T[j] ← k
5.          return j
6.     else i ← i+1
7. until i = m
8. error "hash table overflow"
```

```
Hash-Search(T, k)
1. i ← 0
2. repeat j ← h(k,i)
3.     if T[j] = k
4.     then return j
5.     i ← i+1
6. until T[j] = NIL or i = m
7. return NIL
```

Thuật toán cơ bản để tìm khóa k

Phương pháp địa chỉ mở

- Các phần tử chỉ lưu trong bảng T, không dùng thêm bộ nhớ mở rộng như phương pháp nối kết.
- "Open addressing" mang ý nghĩa là địa chỉ (address) của phần tử không phải chỉ được xác định bằng "duy nhất" hash value của phần tử đó, mà còn có sự can thiệp của phép "dò tìm" (probing).
- Các phương pháp dò tìm phổ biến:
  - Phương pháp dò tuần tự (Linear probing)
  - Phương pháp dò bậc hai (Quadratic probing)
  - Phương pháp băm kép (Double hashing)

Phương pháp dò tuần tự

- Hàm băm h cho biết địa chỉ của khóa k tại lần thử thứ i

$$\boldsymbol{h(k,i) = (h(k) + i) \bmod m}$$

  - i: thứ tự của lần thử ($i = 0, 1, 2, ...$)
  - $h(k)$: hàm băm
  - m: số slot của bảng băm

Phương pháp dò bậc hai

- Hàm băm h cho biết địa chỉ của khóa k tại lần thử thứ i

$$\boldsymbol{h(k,i) = (h(k) + i^2) \bmod m}$$

  - i: thứ tự của lần thử ($i = 0, 1, 2, ...$)
  - $h(k)$: hàm băm
  - m: số slot của bảng băm

Phương pháp băm kép

- Hàm băm h cho biết địa chỉ của khóa k tại lần thử thứ i

$$\boldsymbol{h(k,i) = (h(k) + i * h'(k)) \bmod m}$$

  - i: thứ tự của lần thử ($i = 0, 1, 2, ...$)
  - $h(k)$ và $h'(k)$: các hàm băm
  - m: số slot của bảng băm

Ưu thế của Phương pháp nối kết

- Đơn giản khi cài đặt
- Sử dụng các cấu trúc dữ liệu cơ bản
- Địa chỉ mở giải quyết được đụng độ nhưng lại có thể gây ra những đụng độ mới
- Nối kết không bị ảnh hưởng về tốc độ khi bảng gần đầy
- Ít tốn bộ nhớ khi bảng thưa (chứa ít phần tử)

Bài tập rèn luyện

- Cho bảng băm T có $m = 11$ slot, hàm băm $h(k) = k \bmod m$
- Cho một dãy phần tử theo thứ tự như sau:

10, 22, 31, 4, 15, 28, 17, 88, 59

- Hãy trình bày kết quả khi thêm các phần tử trên vào bảng băm, với lần lượt từng phương pháp xử lý đụng độ:
  - Nối kết (Chaining)
  - Dò tuần tự (Linear probing)
  - Dò bậc hai (Quadratic probing)
  - Băm kép (Double hashing), với $h'(k) = 1 + (k \bmod (m - 1))$

Bài tập rèn luyện

- Cài đặt bảng băm sử dụng cấu trúc dữ liệu mảng hoặc danh sách liên kết.
- Cài đặt phương pháp xử lý đụng độ nối kết
- Cài đặt phương pháp xử lý đụng độ địa chỉ mở với phương pháp dò tìm tuần tự, bậc hai và băm kép.

THE END