Chương III. Phân tích Thiết kế Giải thuật

Phạm Nguyên Khang
BM. Khoa học máy tính
Khoa CNTT – Đại học Cần Thơ
pnkhang@cit.ctu.edu.vn

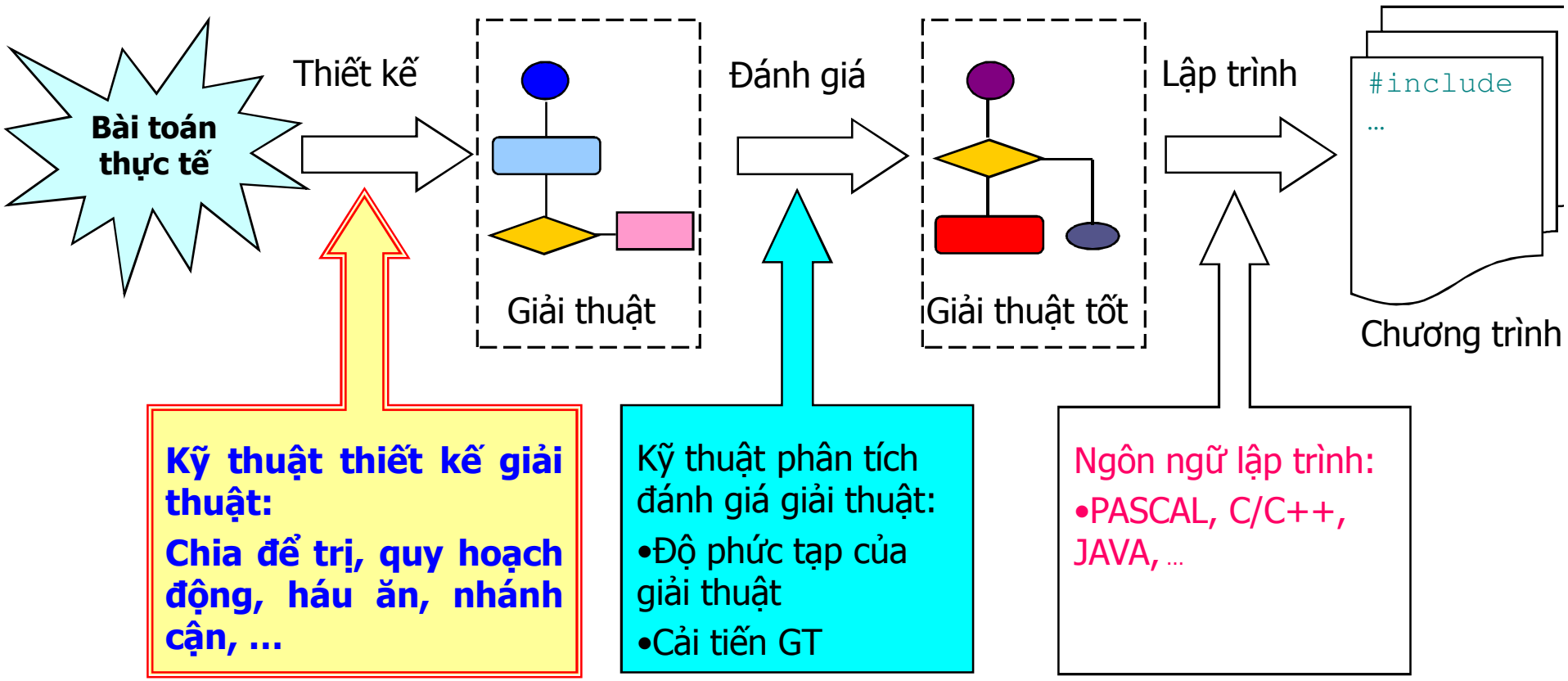
Nội dung

- Mục tiêu
- Từ bài toán đến chương trình
- Các kỹ thuật thiết kế giải thuật
  - Chia để trị
  - Quay lui
    - Vét cạn
    - Nhánh cận
  - Háu ăn/Tham ăn/Tham lam/… (Greedy)
  - Quy hoạch động
- Bài tập

Mục tiêu

- Biết các kỹ thuật thiết kế giải thuật: từ ý tưởng cho đến giải thuật chi tiết.
- Hiểu rõ nguyên lý của các kỹ thuật phân tích thiết kế giải thuật.
- Vận dụng kỹ thuật phân tích thiết kế để giải các bài toán thực tế: các bài toán dạng nào thì có thể áp dụng được kỹ thuật này.

Từ bài toán đến chương trình



Kỹ thuật chia để trị (ý tưởng)

- Yêu cầu:
  - Cần phải giải bài toán có kích thước n.
- Phương pháp:
  1. Ta chia bài toán ban đầu thành một số bài toán con đồng dạng với bài toán ban đầu có kích thước nhỏ hơn n.
  2. Giải các bài toán con được các lời giải con
  3. Tổng hợp lời giải con ➔ lời giải của bài toán ban đầu.
- Chú ý:
  - Đối với từng bài toán con, ta lại chia chúng thành các bài toán con nhỏ hơn nữa.
  - Quá trình phân chia này sẽ dừng lại khi kích thước bài toán **đủ nhỏ** mà ta có thể giải dễ dàng ➔ Gọi là bài toán cơ sở.

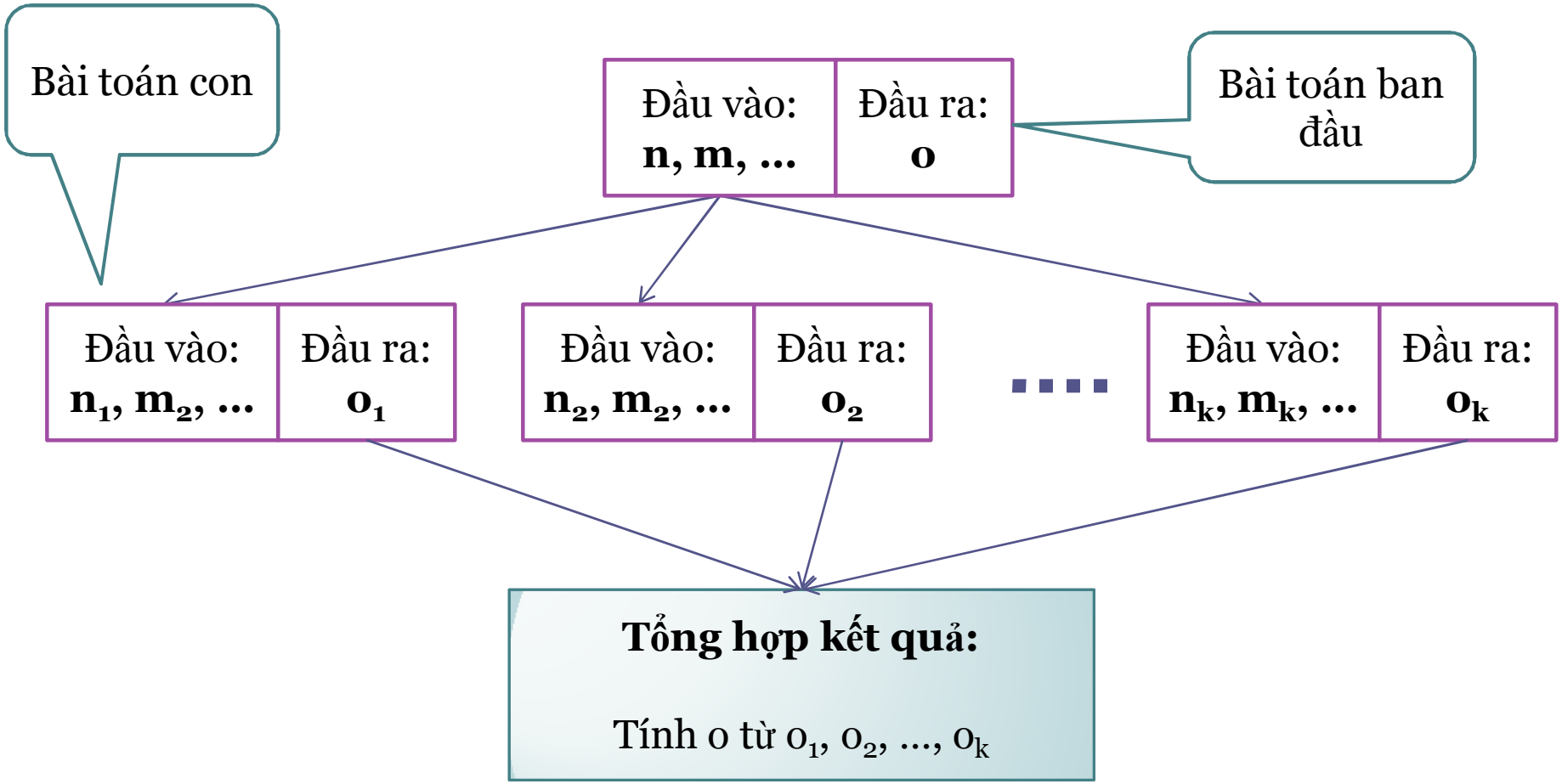
Ví dụ: Quick sort

- Giải thuật Quick Sort
  - Sắp xếp dãy n số theo thứ tự tăng dần
- Áp dụng kỹ thuật chia để trị:
  1. Chia dãy n số thành 2 dãy con
     - Trước khi chia phải phân hoạch
  2. Giải 2 bài toán con
     - Sắp xếp dãy bên trái
     - Sắp xếp dãy bên phải
  3. Tổng hợp kết quả:
     - Không cần tổng hợp

Ví dụ: Merge Sort

- Giải thuật Merge Sort
  - Sắp xếp dãy n số theo thứ tự tăng dần
- Áp dụng kỹ thuật chia để trị:
  1. Chia dãy n số thành 2 dãy con
     - Không cần phân hoạch, cứ cắt dãy số ra làm 2
  2. Giải 2 bài toán con
     - Sắp xếp dãy bên trái ➔ KQ1
     - Sắp xếp dãy bên phải ➔ KQ2
  3. Tổng hợp kết quả:
     - Trộn kết quả của 2 bài toán con

Kỹ thuật chia để trị (phân tích)



Kỹ thuật chia để trị (giải thuật)

```
solve(n) {
    if (n đủ nhỏ để có thể giải được)
        giải bài toán ➔ KQ
        return KQ;
    else {
        Chia bài toán thành các bài toán con
                                kích thước n1, n2, …

        KQ1 = solve(n1); //giải bài toán con 1
        KQ2 = solve(n2); //giải bài toán con 2
        …
        Tổng hợp các kết quả KQ1, KQ2, … ➔ KQ
        return KQ;
}
```

Bài tập: Tìm phần tử trội

- Cho mảng *n* phần tử
- Phần tử trội: phần tử xuất hiện nhiều hơn n/2 lần
- Tìm phần tử trội của 1 mảng n phần tử. Các phần tử chỉ có thể so sánh == hoặc !=
- Gợi ý:
  - Chia mảng thành 2 mảng con

Giảm để trị

- Trường hợp đặc biệt của chia để trị
- Áp dụng cho các bài toán tìm kiếm
  - Tìm điểm chia cắt
  - Tùy theo điều kiện (ví dụ: =, <, >) mà chọn phần xử lý phù hợp
- Chú ý:
  - Quá trình chia cắt sẽ dừng khi không còn gì để chia
  - Phải kiểm tra điều kiện trước khi chia cắt

Ví dụ

- Tìm kiếm nhị phân trên một dãy đã sắp xếp
  - Tìm phần tử có giá trị x trong mảng n phần tử. Phần tử đầu tiên có vị trí 1. Trả về vị trí tìm thấy, nếy không tìm thấy trả về 0
- Kỹ thuật giảm để trị
  - Tìm phần tử giữa
  - So sánh x với phần tử giữa
  - Nếu bằng nhau ➔ Trả về vị trí giữa
  - Nếu x nhỏ hơn ➔ Tìm nửa trái
  - Nếu x lớn hơn ➔ Tìm nửa phải
  - Trả về 0

Kỹ thuật quay lui (ý tưởng)

- Giải bài toán tối ưu
  - Tìm một **lời giải tối ưu** trong số các lời giải
  - Mỗi **lời giải** gồm thành *n* **thành phần**.
  - Quá trình xây dựng một lời giải được xem như quá trình tìm *n* thành phần. Mỗi thành phần được tìm kiếm trong 1 bước.
    - Các **bước** phải có **dạng giống nhau**.
    - Ở mỗi bước, ta có thể dễ dàng chọn lựa một thành phần.
  - Sau khi thực hiện đủ *n* bước ta được 1 lời giải

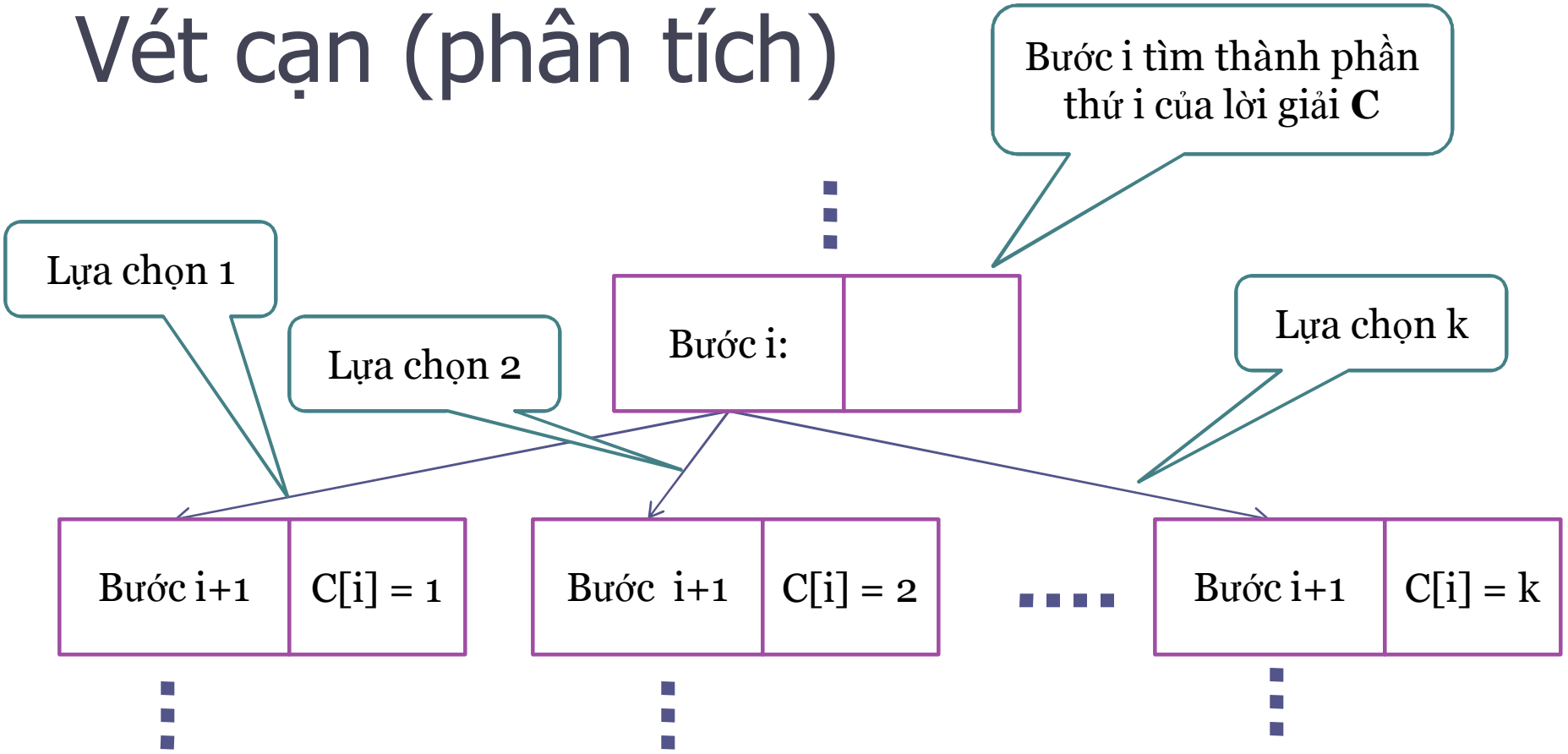
Kỹ thuật quay lui (phương pháp)

- Phương pháp
  - Vét cạn (brute force)
    - Tìm hết tất cả các lời giải
    - Độ phức tạp thời gian lũy thừa
  - Nhánh cận (branch and bound)
    - Chỉ tìm những lời giải có lợi
    - Cải tiến thời gian thực hiện

Vét cạn (ý tưởng)

- Ý tưởng:
  - Gần giống chia để trị nhưng xây dựng lời giải từ dưới lên trong khi chia để trị là phân tích từ trên xuống
- Một phương án/lời giải C:
  - Gồm *n* thành phần C[1], C[2], …, C[n]
- Ở mỗi bước i, có một số lựa chọn cho thành phần i.
  1. Chọn một giá trị nào đó cho thành phần i
  2. Gọi đệ quy để tìm thành phần i + 1
  3. Hủy bỏ sự lựa chọn, quay lui lại bước 1 chọn giá trị khác cho thành phần i
- Chú ý:
  - Quá trình đệ quy kết thúc khi i > n
  - Khi tìm được lời giải, so sánh với các lời trước đó để chọn lời giải tối ưu

Vét cạn (phân tích)



Vét cạn (giải thuật)

```
search(int i) {
   if (i > n)
      Kiem tra, so sánh lời giải với các
      lời giải hiện có ➔ Lời giải tối ưu
   else {
      for (j ∈ lựa chọn có thể có của bước i) {
         C[i] = j;  //Lựa chọn p/a j cho bước i
         search(i + 1); //Gọi đệ quy
         C[i] = null; //Hủy bỏ lựa chọn
      }
   }
}
```

Ví dụ: Bài toán cái ba lô

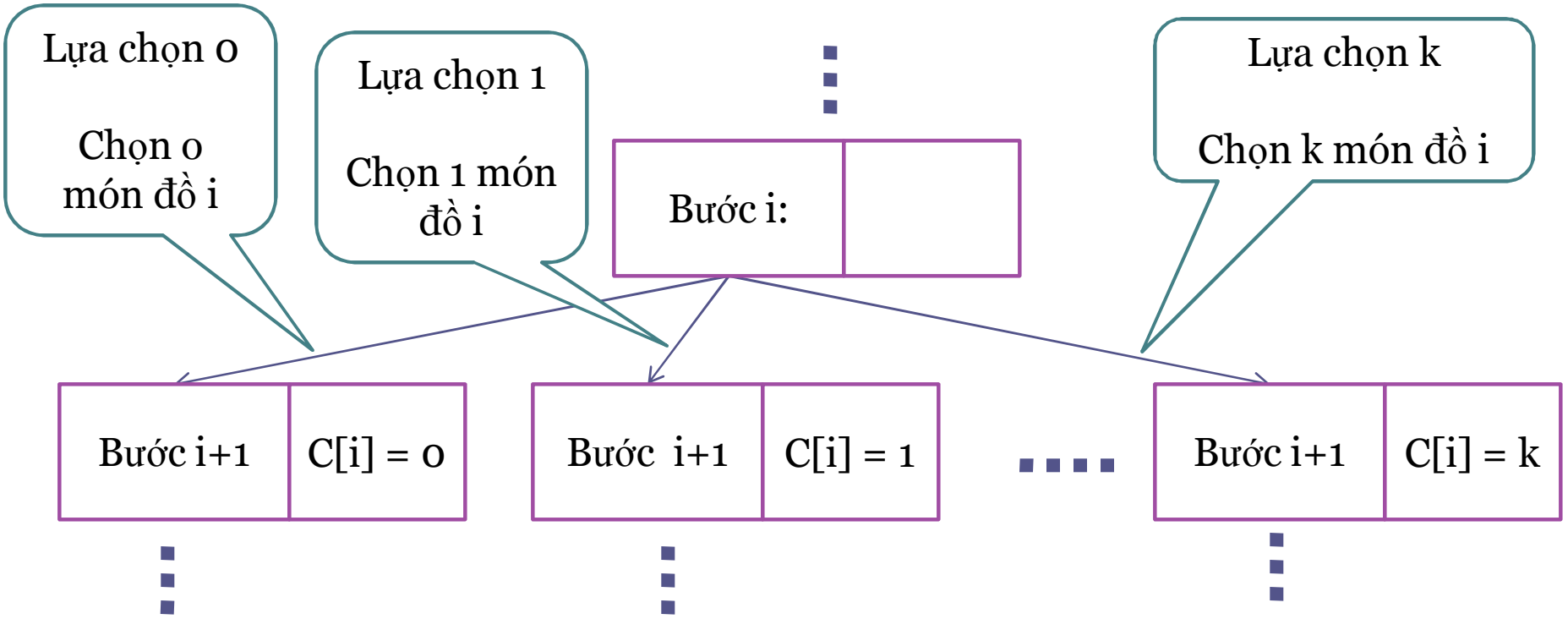
- Có *n* món đồ, món đồ i có
  - giá trị là **v[i]** ($)
  - khối lượng là **g[i]** (kg)
- Số lượng các món đồ là vô hạn
- Có một cái balô có sức chứa tối đa M (kg).

- Yêu cầu:
  - Chọn các món đồ cho vào ba lô **sao cho tổng giá trị lớn nhất và không vượt quá sức chứa của cái ba lô**.

Bài toán cái ba lô (vét cạn)

- Các biến:
  - V_opt: tổng giá trị tối ưu
  - C_opt: phương án tối ưu
  - V, C: tổng giá trị và phương án hiện hành
- Ở mỗi bước i:
  - Giả sử ta đã có ở bước i - 1:
    - V: tổng giá trị các món đồ đã được chọn
    - G: tổng khối lượng các món đồ đã được chọn
  - Chỉ có thể lựa chọn 0, 1, 2, …, hoặc n_max món đồ i
    - Với **n_max = (M - G)/g[i]**
  - Với mỗi lựa chọn j cho bước i
    - Lựa chọn j cho bước i, cập nhật lại V, G (thêm vào)
    - Gọi đệ quy cho bước i + 1
    - Hủy bỏ sự lựa chọn j, cập nhật lại V, G (bớt ra)

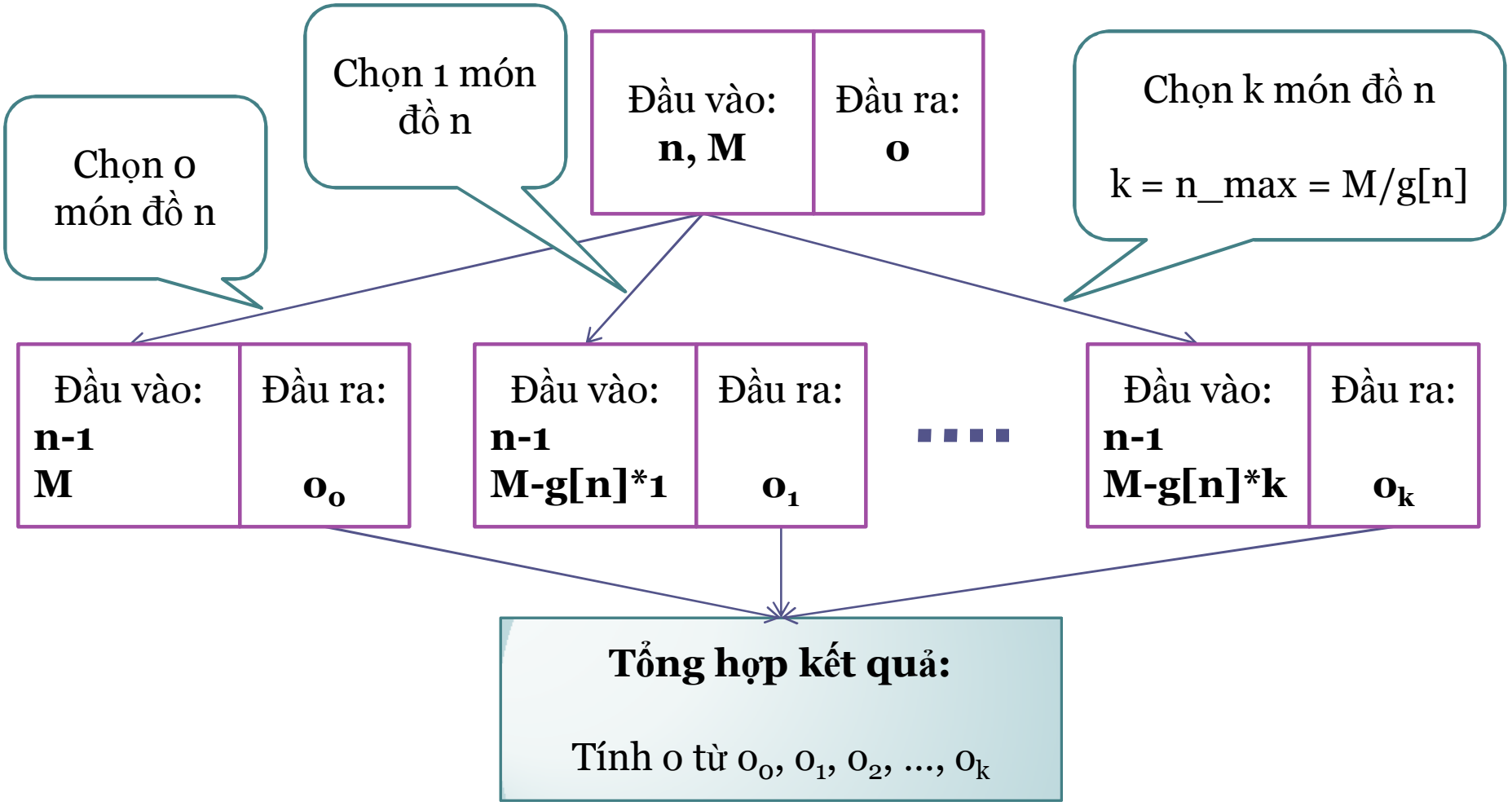
Bài toán cái ba lô (vét cạn)



Bài toán cái ba lô (vét cạn)

```
search (int i) {
   if (i > n) {
      if (V > V_opt) {
        copy c cho c_opt ➔ Trien khai tiep bang vong lap for
        V_opt = V;
      }
   } else {
      int N_max = (M - G)/g[i]; //chỉ có thể chọn tối đa N_max
      for (int j = 0; j <= N_max; j++) {
        c[i] = j;  //Đánh dấu lựa chọn
        G    = G + g[i]*c[i]; V = V + v[i]*c[i];
        search (i + 1); //Gọi đệ quy
        G    = G - g[i]*c[i]; V = V - v[i]*c[i];
      }
   }
}
```

Bài toán cái ba lô (chia để trị)



Bài toán cái ba lô (chia để trị)

```
int sovle(int M, int n) {
    if (n == 1) {
        return v[n]*(M/g[n]);
    } else {
        int n_max = M/g[n];
        int V_opt = -1;
        for (int i = 0; i <= n_max; i++) {
            int V = i*v[n] + solve(M - i*g[n], n - 1);
            if (V > V_opt)
                V_opt = V;
        }
        return V_opt;
    }
}
```

- Chỉ tìm tổng giá trị lớn nhất
- Không tìm phương án chọn các món đồ như thế nào để đạt tổng giá trị lớn nhất
- **Tuy nhiên có thể cải tiến khá dễ dàng giải thuật để có được phương án lựa chọn tối ưu**

Nhánh cận

- Cải tiến giải thuật quay lui vét cạn
  - Tại mỗi bước, ta sẽ xem xét xem có nên đi bước kế tiếp nữa hay không
  - Việc xem xét dựa trên **khái niệm cận** của bước hiện hành

Vét cạn vs Nhánh cận

Vét cạn

```
else {
  for (j ∈ LC của i){
  C[i] = j;
  search(i + 1);
  C[i] = null;
  }
}
```

Nhánh cận

```
else {
  for (j ∈ LC của i)
     tính cận cho LC j
  S. xếp các LC theo cận
  for (j ∈ LC của i) {
    if (cận của j còn tốt) {
      c[i] = j;
      search (i + 1);
      C[i] = null;
    }
  }
}
```

Kỹ thuật háu ăn (greedy)

- Mục đích:
  - Tìm một lời giải tốt trong thời gian chấp nhận được (độ phức tạp đa thức thay vì lũy thừa)
- Ý tưởng
  - Chia quá trình tìm lời giải thành nhiều bước như kỹ thuật quay lui
- Với mỗi bước
  - Sắp xếp các lựa chọn cho bước đó theo thứ tự nào đó "có lợi" (tăng dần hoặc giảm dần tùy theo cách lập luận)
  - Chọn lựa chọn tốt nhất rồi đi tiếp bước kế (**không quay lui**)

Quy hoạch động

- Mục đích:
  - Cải tiến thuật toán chia để trị hoặc quay lui vét cạn để giảm thời gian thực hiện
- Ý tưởng:
  - Lưu trữ các kết quả của các bài toán con trong BẢNG QUY HOẠCH (cơ chế **caching**)
  - Đổi **bộ nhớ** lấy **thời gian** (trade memory for time)

- Thiết kế giải thuật bằng kỹ thuật Quy hoạch động
  1. Phân tích bài toán dùng kỹ thuật chia để trị/quay lui
     - Chia bài toán thành các bài toán con
     - **Tìm quan hệ giữa KQ của bài toán lớn và KQ của các bài toán con (công thức truy hồi)**
  2. Lập bảng quy hoạch

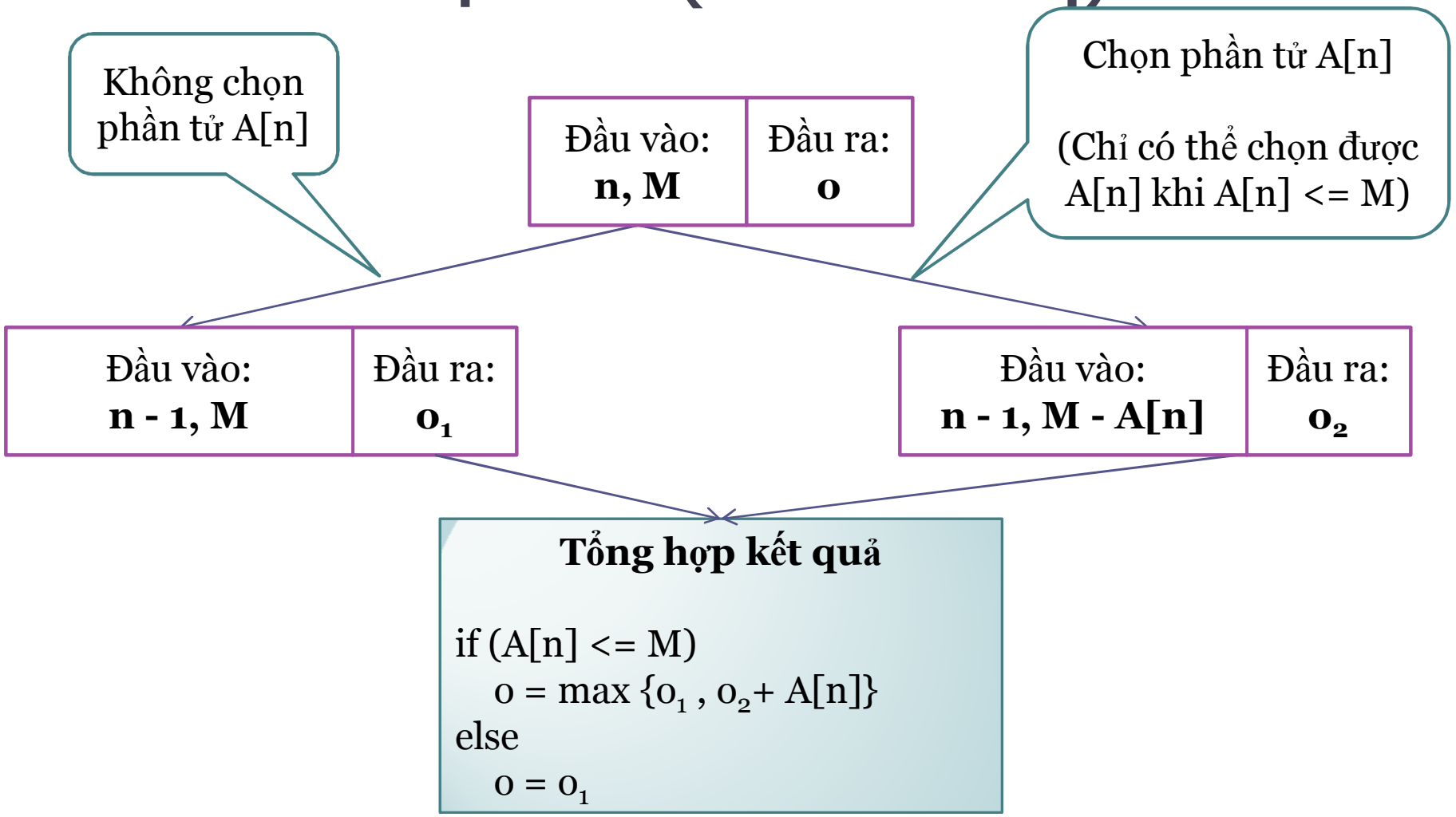
Quy hoạch động

- Lập bảng quy hoạch
  - Số chiều = số biến trong công thức truy hồi
  - Thiết lập quy tắc điền kết quả vào bảng quy hoạch
    - Điền các ô không phụ thuộc trước
    - Điền các ô phụ thuộc sau
  - Tra bảng tìm kết quả (thường chỉ tìm được giá trị)
  - Lần vết trên bảng để tìm lời giải tối ưu

Ví dụ

- Bài toán chọn số (tương tự bài toán cái ba lô):
  - Cho A là một mảng n số nguyên dương
  - Chọn một số phần tử của mảng A sao cho tổng lớn nhất nhưng không được vượt quá M
  - Tìm tổng lớn nhất này

Bài toán chọn số (chia để trị)



Bài toán chọn số (chia để trị)

```
int solve (int n, int M) {
  if (n == 0)
    return 0; //không có gì để chọn
  else {
    int o1 = solve(n - 1, M);
    if (A[n] <= M) {
      int o2 = solve(n - 1, M - A[n]);
      return max(o1, o2 + A[n]);
    } else
      return o1;
  }
}
```

Bài toán chọn số (quy hoạch động)

- Sử dụng kỹ thuật chia để trị phân tích bài toán
- Tìm mối quan hệ giữa KQ của bài toán lớn và KQ của các bài toán con (công thức truy hồi)

Tổng hợp kết quả

if (A[n] <= M)
    o = max {o_1 , o_2+ A[n]}
else
    o = o_1

Công thức truy hồi

F(n, M) = max {F(n-1, M) ,
F(n-1, M - A[n]}, *nếu A[n] <= M*

F(n, M) = F(n-1, M) *ngược lại*

Bài toán chọn số (quy hoạch động)

- Xây dựng bảng quy hoạch
- Hàm F(n, M) có **2** tham số
  - Bảng quy hoạch là một mảng **2** chiều
- Điền kết quả vào bảng quy hoạch
  - Điền trường hợp suy biến/cơ sở (n = 0) trước
    - Toàn bộ hàng 0 có giá trị 0
  - Điền các trường hợp khác sau (từ 1 → n)
    - Ô F(i, j) được tính theo công thức truy hồi

Bài toán chọn số (quy hoạch động)

```
int solve (int n, int M) {
    int F[MAX][MAX];
    //hàng 0
    for (int j = 0; j <= M; j++)
        F[0][j] = 0;
    //hàng từ 1 đến n
    for (int i = 1; i <= n; i++)
        for (int j = 0; j <= M; j++)
        if (A[i] <= j)
            F[i][j] = max(F[i-1][j],
                          F[i-1][j-A[i]] + A[i]);
        else
            F[i][j] = F[i-1][j];
    return F[n][M];
}
```

Chia để trị vs Quy hoạch động

Chia để trị

- Ý tưởng
  - Phân rã thành các bài toán con
  - Tổng hợp kết quả
- Giải thuật:
  - Đệ quy từ trên xuống
  - Độ phức tạp thời gian lớn nếu có nhiều bài con giống nhau
  - **Không cần lưu trữ kết quả của tất cả các bài toán con**

Quy hoạch động

- Ý tưởng:
  - Phân rã thành các bài toán con
  - Tìm mối quan hệ
- Giải thuật:
  - Lập bảng quy hoạch và giải từ dưới lên
  - **Độ phức tạp thời gian nhỏ hơn nhờ sử dụng bảng quy hoạch**
  - Cần bộ nhớ để lưu trữ bảng quy hoạch

Kết hợp Quy hoạch động và đệ quy

- Sử dụng bảng quy hoạch để lưu kết quả bài toán con
- Không cần điền hết tất cả bảng quy hoạch
  - Điền bảng quy hoạch theo yêu cầu
    - Bắt đầu từ bài toán gốc
    - Nếu trong bảng quy hoạch chưa có KQ, gọi đệ quy để tìm kết quả và **lưu kết quả vào bảng quy hoạch**
    - Nếu KQ đã có trong bảng quy hoạch, sử dụng ngay kết quả này
- Có thể sử dụng bảng băm để lưu trữ bảng quy hoạch

Kết luận

- Mỗi kỹ thuật chỉ phù hợp với 1 hoặc 1 số loại bài toán
- Mỗi kỹ thuật đều có ưu và khuyết điểm, không có kỹ thuật nào là "trị bá bệnh"
  - Kỹ thuật nhánh cận cần phải có cách ước lượng cận tốt mới mong cắt được nhiều nhánh
  - Quy hoạch động chỉ tốt khi số lượng bài toán con cần phải giải là đa thức (n, n^2 hoặc n^3)
  - Kỹ thuật vét cạn có độ phức tạp thời gian quá cao (lũy thừa)
    - Chỉ dùng khi n nhỏ hoặc khi không còn cách nào khác