THUẬT TOÁN ỨNG DỤNG

ĐỆ QUY QUAY LUI

Phạm Quang Dũng
Bộ môn KHMT
dungpq@soict.hust.edu.vn

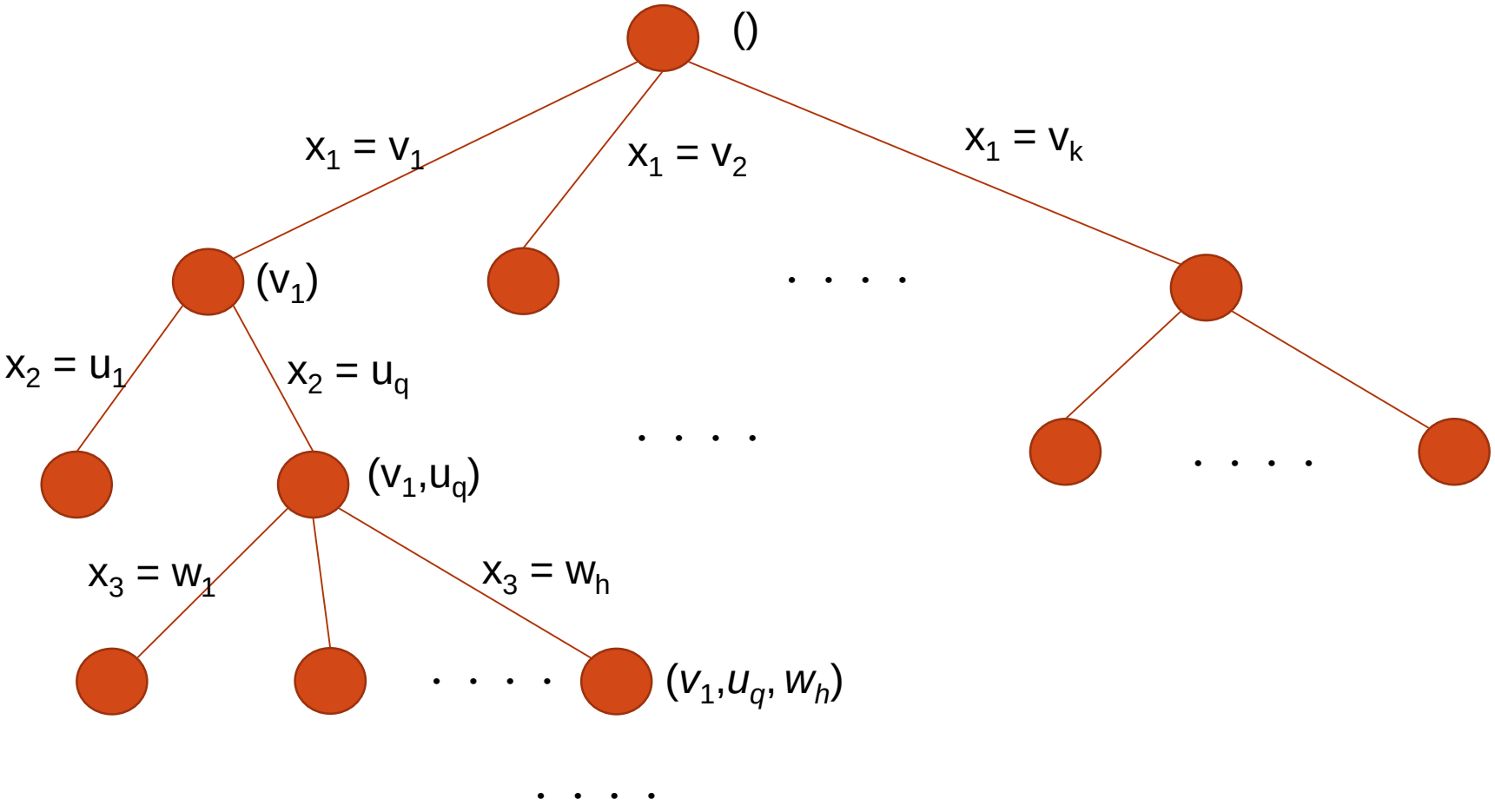
Nội dung

- Đệ quy quay lui
  - Tổng quan đệ quy quay lui
  - Bài toán liệt kê xâu nhị phân
  - Bài toán liệt kê nghiệm nguyên dương phương trình tuyến tính
  - Bài toán liệt kê TSP
  - Bài toán liệt kê hành trình taxi
  - Bài toán liệt kê CBUS
  - Bài toán liệt kê BCA
  - Bài toán liệt kê CVRP
- Thuật toán nhánh và cận
  - Tổng quan nhánh và cận
  - Bài toán tối ưu TSP
  - Bài toán tối ưu hành trình taxi
  - Bài toán tối ưu CBUS
  - Bài toán tối ưu BCA
  - Bài toán tối ưu CVRP

Đệ quy quay lui

- Phương pháp dùng để liệt kê các cấu hình tổ hợp cũng như giải bài toán tối ưu tổ hợp
  - Liệt kê: liệt kê tất cả các bộ $x = (x_1,x_2,\ldots, x_N)$ trong đó $x_i \in A_i$ - tập rời rạc, đồng thời $(x_1,x_2,..., x_N)$ thỏa mãn các ràng buộc C cho trước
  - Tối ưu tổ hợp: trong số các bộ (phương án) $x = (x_1,x_2,\ldots, x_N)$ trong đó $x_i \in A_i$ - tập rời rạc, đồng thời $(x_1,x_2,..., x_N)$ thỏa mãn các ràng buộc C cho trước, cần tìm phương án có $f(x) \rightarrow$ min(max)
- Thử lần lượt từng giá trị cho mỗi biến
  - Chiến lược chọn biến, ví dụ $x_1, x_2, x_3,\ldots, x_N$
  - Chiến lược chọn giá trị cho biến, ví dụ từ nhỏ đến lớn hoặc ngược lại

Đệ quy quay lui



Đệ quy quay lui

- Tại mỗi thời điểm, ta có phương án bộ phận ($x_1 = v_1$, $x_2 = v_2$, …, $x_{k-1} = v_{k-1}$) → cần thử duyệt tiếp các khả năng cho x_k ?

```
try(k){// thử giá trị cho x_k
 forall v ∈ A_k do{
  if(check(v,k)) then{// kiểm tra ràng buộc C của bài toán
    x_k = v;
    [cập nhật các cấu trúc dữ liệu liên quan]
    if(k = N) then solution();
    else try(k+1);
    [khôi phục trạng thái các cấu trúc dữ liệu khi quay lui]
  }
 }
}
```

Liệt kê xâu nhị phân độ dài N

```
#include <iostream>
using namespace std;
#define MAX 100
int N;
int x[MAX];// bieu dien loi phuong an cua bai toan

bool check(int v, int k){
    return true;
}
void solution(){
    for(int i = 1; i <= N; i++) cout << x[i] << " "; cout << endl;
}
```

Liệt kê xâu nhị phân độ dài N

```
void TRY(int k){
    for(int v = 0; v <= 1; v++){
        if(check(v,k)){
            x[k] = v;
            if(k == N) solution();
            else TRY(k+1);
        }
    }
}
int main(){
    N = 3;
    TRY(1);// bat dau thu gia tri cho x[1]
}
```

Liệt kê nghiệm nguyên dương của phương trình tuyến tính

$$X_1 + X_2 + \ldots + X_n = M$$

- Phương án bộ phận (X_1, …, X_{k-1}) có tổng bộ phận là T
- Khởi tạo
  - Phương án bộ phận rỗng: (), T = 0
- Thử giá trị cho X_k
  - $X_1 + X_2 + \ldots + X_{k-1} + X_k + X_{k+1} + \ldots + X_n = M$
  - $X_k = M - T - (X_{k+1} + X_{k+2} + \ldots + X_n)$
  - $1 \le X_k \le M - T - (n - k)$

Liệt kê nghiệm nguyên dương của phương trình tuyến tính

$$X_1 + X_2 + \ldots + X_n = M$$

```
#include <stdio.h>
#define MAX 100

int n,M;
int X[MAX];
int T;// accumulated sum

int check(int v, int k){
    if(k < n) return 1;
    return T + v == M;
}
void solution(){
    for(int i = 1; i <= n; i++) printf("%d ",X[i]); printf("\n");
}
```

Liệt kê nghiệm nguyên dương của phương trình tuyến tính

```
void TRY(int k){
    for(int  v = 1; v <= M-T - n+k; v++){
        if(check(v,k)){
            X[k] = v;
            T = T + X[k];// update incrementally
            if(k == n) solution();
            else TRY(k+1);
            T = T - X[k];// recover when backtracking
        }
    }
}
int main(){
    n = 6; M = 15;
    T = 0;
    TRY(1);
}
```

Liệt kê hành trình giao hàng

- Một shipper nhận hàng từ cửa hàng (điểm 0) và phải đi qua tất cả *N* khách hàng 1, 2, 3, . . ., *N* (mỗi khách hàng đi qua đúng 1 lần) để giao hàng. Hãy liệt kê tất cả các phương án cho shipper

Liệt kê hành trình giao hàng

```
#include <bits/stdc++.h>
#define MAX 100
using namespace std;

int N;
int X[MAX];// permutation 1,2,...,N
int appear[MAX];// appear[v] = 1 indicates that v has appeared

void solution(){
   for(int i = 1; i <= N; i++) cout << X[i] << " "; cout << endl;
}
bool check(int v, int k){
    return appear[v] == 0;
}
```

Liệt kê hành trình giao hàng

```
void TRY(int k){// thu gia tri cho X[k]
    for(int v = 1; v <= N; v++){
        if(check(v,k)){
            X[k] = v;
            appear[v] = 1;// update
            if(k == N) solution();
            else TRY(k+1);
            appear[v] = 0;// recover
        }
    }
}
int main(){
    N = 3;
    for(int v = 1; v <= N; v++) appear[v] = 0;
    TRY(1);
}
```

Liệt kê hành trình đón trả khách cho taxi

- Một taxi xuất phát từ điểm 0 và cần phục vụ đón trả *N* khách 1, 2, …, *N*. Khách thứ i có điểm đón là *i* và điểm trả là *N+i*. Hãy liệt kê tất cả các phương án đón trả cho taxi.

- Giải pháp
  - Đưa về bài toán liệt kê hoán vị
  - Do tính chất vận chuyển taxi nên điểm *i* sẽ nằm ngay trước điểm *i+N* trên mỗi hoán vị của 1, 2, …, 2*N*

  → mỗi hoán vị của 1, 2, …, *N*, chèn thêm *i+N* vào ngay sau giá trị *i (với mọi i = 1, 2, …, N)*

Liệt kê hành trình đón trả khách cho taxi

```
#include <bits/stdc++.h>
#define MAX 100
using namespace std;

int N;
int X[MAX];// permutation 1,2,...,N
int appear[MAX];// appear[v] = 1 indicates that v has appeared

void solution(){
  for(int i = 1; i <= N; i++)
    cout << X[i] << " " << X[i] + N << "  ";
  cout << endl;
}
bool check(int v, int k){
    return appear[v] == 0;
}
```

Liệt kê hành trình đón trả khách cho taxi

```
void TRY(int k){// thu gia tri cho X[k]
    for(int v = 1; v <= N; v++){
        if(check(v,k)){
            X[k] = v;
            appear[v] = 1;// update
            if(k == N) solution();
            else TRY(k+1);
            appear[v] = 0;// recover
        }
    }
}
int main(){
    N = 3;
    for(int v = 1; v <= N; v++) appear[v] = 0;
    TRY(1);
}
```

Liệt kê hành trình đón trả khách cho bus

- Một xe bus xuất phát từ điểm 0 và cần phục vụ đón trả *N* khách 1, 2, …, *N*. Khách thứ i có điểm đón là *i* và điểm trả là *N+i*. Xe bus chở được tối đa *Q* khách cùng một lúc. Hãy liệt kê tất cả các phương án đón trả cho xe bus.

Liệt kê hành trình đón trả khách cho bus

- Một xe bus xuất phát từ điểm 0 và cần phục vụ đón trả *N* khách 1, 2, …, *N*. Khách thứ i có điểm đón là *i* và điểm trả là *N+i*. Xe bus chở được tối đa *Q* khách cùng một lúc. Hãy liệt kê tất cả các phương án đón trả cho xe bus.

- Thiết kế giải pháp
  - Kiểm soát số hành khách trên xe bằng biến q: số khách đang có mặt trên xe
    - Mỗi khi hành trình đi qua 1 điểm đón thì tăng q lên 1
    - Mỗi khi hành trình đi qua 1 điểm trả thì giảm q đi 1 đơn vị

Liệt kê hành trình đón trả khách cho bus

```
#include <bits/stdc++.h>
using namespace std;
#define MAX_N 100
int N;// so khach
int Q;// so cho tren bus cho hanh khach
int X[2*MAX_N + 1];// bieu dien phuong an lo trinh X[1], X[2], . . . X[2N]
int q;// so khach thuc su dang co tren xe ung voi phuong an bo phan hien tai
bool appear[2*MAX_N+1];
bool check(int v, int k){
    if(appear[v]) return false;
    if(v <= N){// v is pickup
        if(q >= Q) return false;
    }else{// v > N means drop-off
        if(!appear[v-N]) return false;
    }
    return true;
}
```

Liệt kê hành trình đón trả khách cho bus

```
void solution(){
    for(int i = 1; i <= 2*N; i++) cout << X[i] << " "; cout << endl;
}
void TRY(int k){// thu gia tri cho X[k]
    for(int v = 1; v <= 2*N; v++){
        if(check(v,k)){
            X[k] = v;
            appear[v] = true;
            if(v <= N) q++; else q--;// update q incrementally
            if(k == 2*N) solution();
            else TRY(k+1);
            appear[v] = false;
            if(v <= N) q--; else q++;// recover status q
        }
    }
}
```

Liệt kê hành trình đón trả khách cho bus

```
int main(){
    N = 3; Q = 2;
    q = 0;
    for(int v = 1; v <= 2*N; v++) appear[v] = false;
    TRY(1);
}
```

DIGITS

- Write C program that reads an integer value ***N*** from stdin, prints to stdout the number Q ways to assign values 1, 2, …, 9 to characters I, C, T, H, U, S, K (characters are assigned with different values)

ICT – K62 + HUST = N

stdin	stdout
1234	24

DIGITS

```
#include <stdio.h>

int X[7];// X[0] = I, X[1] = C, X[2] = T, X[4] = H, X[5] = U, X[6] = S, X[3] = K
int appeared[10];
int ans,N;

void solution(){
    int T = X[0]*100 + X[1]*10 + X[2] - X[3]*100 - 62 +
    X[4]*1000 + X[5]*100 + X[6]*10 + X[2];
    if(T == N){
        ans++;
        //printf("%d%d%d - %d62 + %d%d%d%d\n",X[0],X[1],X[2],X[3],X[4],X[5],X[6],X[2]);
    }
}
void init(){
    for(int v = 1; v <= 9; v++) appeared[v] = 0;
}
```

DIGITS

```
void TRY(int k){
    for(int v = 1; v <= 9; v++){
        if(appeared[v] == 0){
            X[k] = v;
            appeared[v] = 1;
            if(k == 6){
                solution();
            }else{
                TRY(k+1);
            }
            appeared[v] = 0;
        }
    }
}
```

DIGITS

```
void solve(){
    scanf("%d",&N);
    init();
    ans = 0;
    TRY(0);
    printf("%d",ans);
}
int main(){
    solve();
}
```

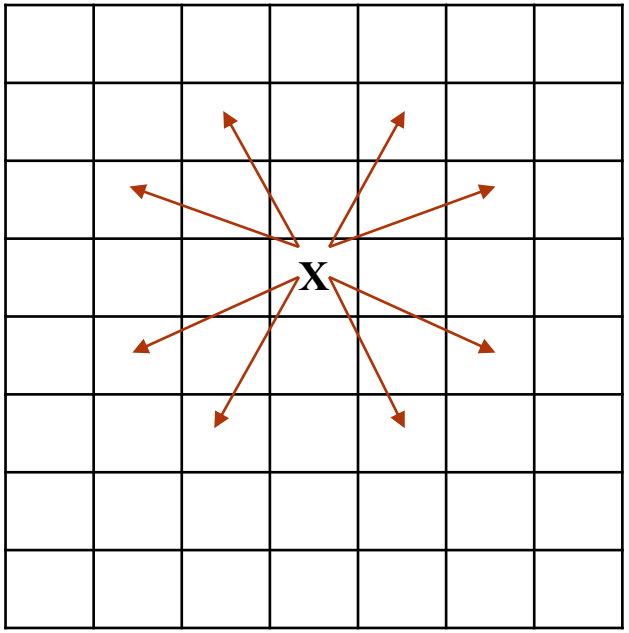
KNIGHT

- Cho một bàn cờ quốc tế N x N. Một quân mã xuất phát từ ô (i, j). Hãy liệt kê tất cả các hành trình cho quân mã di chuyển (theo luật cờ vua) đến tất cả các ô của bàn cờ, mỗi ô đúng 1 lần
- Input
  - Duy nhất 1 dòng chứa N, i, j ($1 \le i, j \le N \le 8$)
- Output
  - Ghi ra dãy tọa độ các ô trên hành trình của quân mã

KNIGHT

- Từ mỗi ô (*i*,*j*), quân mã có thể nhảy đến các ô (*i*+*di*, *j*+*dj*) với (*di*,*dj*) ∈{(1,2), (1,-2), (-1,2), (-1,-2), (2,1), (2,-1),(-2,1),(-2,-1)}
- Biểu diễn phương án
  - Xi[1..N*N], Xj[1..N*N]
  - (Xi[1],Xj[1]) → (Xi[2],Xj[2]) → ..
- Mảng đánh dấu mark[i][j] = true

có nghĩa ô (i,j) đã được đi đến



KNIGHT

```
#include <bits/stdc++.h>
using namespace std;
#define MAX 30
int di[8] = {1,  1, 2,  2, -1, -1, -2, -2};
int dj[8] = {2, -2, 1, -1, 2, -2,   1, -1};

bool mark[MAX][MAX];
int Xi[MAX*MAX];
int Xj[MAX*MAX];

int N;
int I,J; // start cell (I,J)
bool found;
int cnt;
```

KNIGHT

```
bool check(int r, int c){
    if(r < 1 || r > N) return false;
    if(c < 1 || c > N) return false;
    return !mark[r][c];
}
void solution(){
    found = true;
    cnt++;
    for(int k = 1; k <= N*N; k++) cout << "(" << Xi[k] << "," << Xj[k] << ") ";
    cout << endl;
}
```

KNIGHT

```
void TRY(int k){// current cell (Xi[k-1],Xj[k-1])
    for(int q = 0; q < 8; q++){
        if(check(Xi[k-1] + di[q], Xj[k-1] + dj[q])){
            Xi[k] = Xi[k-1] + di[q];
            Xj[k] = Xj[k-1] + dj[q];
            mark[Xi[k]][Xj[k]] = true;// update
            if(k == N*N) solution();
            else TRY(k+1);
            mark[Xi[k]][Xj[k]] = false;// recover
        }
    }
}
```

KNIGHT

```
int main(){
    cin >> N >> I >> J;
    for(int i = 1; i <= N; i++)
        for(int j = 1; j <= N; j++)
            mark[i][j] = false;
    Xi[1] = I; Xj[1] = J;// starting cell
    mark[I][J] = true;
    found = false;
    cnt = 0;
    TRY(2);
    cout << cnt;
}
```

Liệt kê phương án phân công giảng dạy

- Có *n* môn học 1, 2, …, *n* cần được phân cho *m* giáo viên 1, 2, …, *m*. Mỗi giáo viên có danh sách các môn mà người này có thể giảng dạy (tùy thuộc chuyên ngành hẹp của giáo viên).
- Vì đã được xếp thời khóa biểu từ trước nên giữa n môn học này sẽ có các cặp 2 môn trùng thời khóa biểu và do đó không thể được phân công cho cùng một giáo viên được và được thể hiện bởi 0-1 ma trận $A(i,j)$ trong đó $A(i,j) = 1$ có nghĩa môn *i* và *j* trùng thời khóa biểu.
- Hãy liệt kê tất cả các phương án phân công giảng dạy các môn cho giáo viên

Liệt kê phương án phân công giảng dạy

- Dữ liệu đầu vào
  - Dòng 1: n, m ($1 \le m < n \le 10$)
  - Dòng i+1 ($i = 1,\ldots,n$): k, t_1, t_2, …, t_k trong đó k là số giáo viên có thể dạy môn i và t_1, t_2, …, t_k là các giáo viên có thể dạy môn i
  - Dòng i+n+1 ($i = 1,\ldots,n$): chứa các phần tử dòng thứ i của ma trận A

Liệt kê phương án phân công giảng dạy

```
#include <bits/stdc++.h>
#define MAX_N 100
#define MAX_M 30
using namespace std;
// input data structures
int N;// number of course
int M;// number of teachers
int sz[MAX_N];// sz[c] is the number of possible teachers for course c
int t[MAX_N][MAX_M];// t[c][i]: the ith teacher that can teach course c
int h[MAX_N];// h[c] is the number of hours of course c each week
int A[MAX_N][MAX_N];// A[i][j] = 1 indicates that course i and j are
conflict
int f[MAX_M];
int cnt; // number of solutions;
// variables
int X[MAX_N];
```

Liệt kê phương án phân công giảng dạy

```
void input(){
    cin >> N >> M;
    for(int i = 1; i <= N; i++)
        cin >> h[i];
    for(int i = 1; i <= N; i++){
        cin >> sz[i];
        for(int j = 0; j < sz[i]; j++)
            cin >> t[i][j];
    }
    for(int i = 1; i <= N; i++){
        for(int j = 1; j <= N; j++)
            cin >> A[i][j];
    }
}
```

Liệt kê phương án phân công giảng dạy

```
int check(int v, int k){
    for(int i = 1; i <= k-1; i++){
        if(A[i][k] && v == X[i]) return 0;
    }
    return 1;
}
void solution(){
    cnt++;
    cout << "solution " << cnt << endl;
    for(int t = 1; t <= M; t++){
        cout << "course of teacher " << t << ": ";
        for(int i = 1; i <= N; i++) if(X[i] == t) cout << i << ", ";
        cout << " hour = " << f[t] << endl;
    }
    cout << "--------------------" << endl;
}
```

Liệt kê phương án phân công giảng dạy

```
void TRY(int k){
    for(int i = 0; i < sz[k]; i++){
        int v = t[k][i];
        if(check(v,k)){
                X[k] = v;
                f[v] += h[k];
                if(k == N){
                    solution();
                }else{
                    TRY(k+1);
                }
                f[v] -= h[k];
            }

    }
}
```

Liệt kê phương án phân công giảng dạy

```
void solve(){
    for(int i = 1; i <= M; i++) f[i] = 0;
    cnt = 0;
    TRY(1);
 }
int main(){
    input();
    solve();
}
```

Bài toán CVRP

- Một đội gồm *K* xe tải giống nhau cần được phân công để vận chuyển hàng hóa pepsi từ kho trung tâm (điểm 0) đến các điểm giao hàng 1,2,…,N.
- Mỗi xe tải có tải trọng Q (mỗi chuyển chỉ vận chuyển tối đa Q thùng).
- Mỗi điểm giao hàng i có lượng hàng yêu cầu là d[i] thùng
- Cần xây dựng phương án vận chuyển sao cho
  - Mỗi xe đều phải được phân công vận chuyển
  - Mỗi điểm giao chỉ được giao bởi đúng 1 xe
  - Tổng lượng hàng trên xe không vượt quá tải trọng của xe đó
- Cần liệt kê tất cả các phương án vận chuyển

Bài toán CVRP

- Ví dụ N = 3, K = 2

→ có 6 phương án vận chuyển sau

Route[1] = 0 – 1 – 0 Route[2] = 0 – 2 – 3 – 0	Route[1] = 0 – 1 – 2 – 0 Route[2] = 0 – 3 – 0
Route[1] = 0 – 1 – 3 – 0 Route[2] = 0 – 2 – 0	Route[1] = 0 – 2 – 0 Route[2] = 0 – 3 – 1 – 0
Route[1] = 0 – 1 – 0 Route[2] = 0 – 3 – 2 – 0	Route[1] = 0 – 2 – 1 – 0 Route[2] = 0 – 3 – 0

Bài toán CVRP

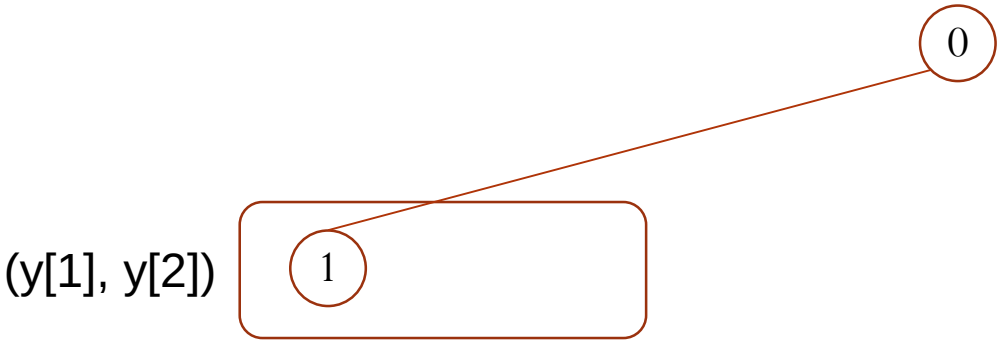
- Thiết kế dữ liệu
  - y[k] điểm giao đầu tiên của xe thứ k (y[k] $\in${1,2,...,N}, với k=1,2,…,K)
  - x[i] là điểm tiếp theo của điểm giao i trên lộ trình (x[i] $\in${0,1,2,…,N}, với i = 1,2,...,N)
  - Do các xe giống hệt nhau nên giả định y[1] < y[2] < … < y[K]
  - visited[v] = true nếu v đã được thăm bởi 1 xe nào đó

Bài toán CVRP

- Chiến lược duyệt
  - Bắt đầu bằng việc duyệt bộ giá trị cho (y[1],. . ., y[K])
  - Với mỗi bộ giá trị đầy đủ của (y[1],. . ., y[K]), bắt đầu duyệt bộ giá trị cho x[1,...,N] xuất phát từ x[y[1]]
  - Mỗi khi thử giá trị x[v] = u cho xe thứ k thì
    - Nếu u > 0 (chưa phải điểm xuất phát) thử duyệt tiếp giá trị cho x[u] vẫn trên chuyến xe thứ k
    - Nếu u = 0 (điểm xuất phát) thì
      - Nếu k = K (đã đủ hết các chuyến cho K) xe và điểm giao nào cũng được thăm thì ghi nhận 1 phương án
      - Ngược lại, thử duyệt tiếp giá trị cho chuyến của xe k+1 bắt đầu bởi cho x[y[k+1]]
  - Biến segments
    - Ghi nhận số chặng (đoạn nối giữa 2 điểm liên tiếp trên đường đi)
    - Khi segments = N+K thì thu được phương án đầy đủ

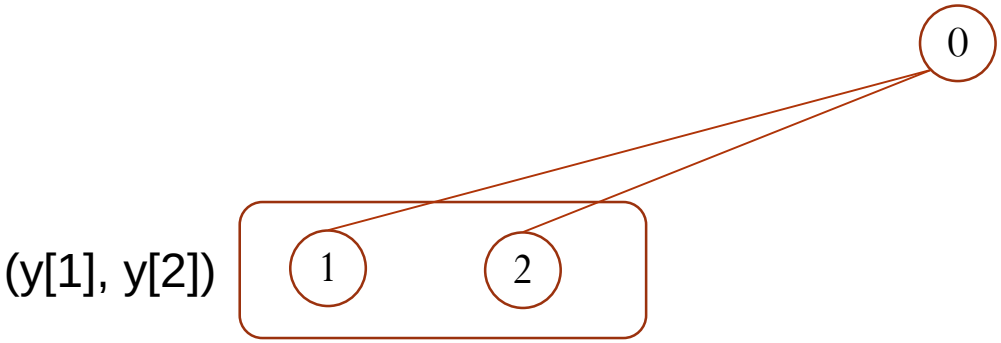
Bài toán CVRP

- K = 2



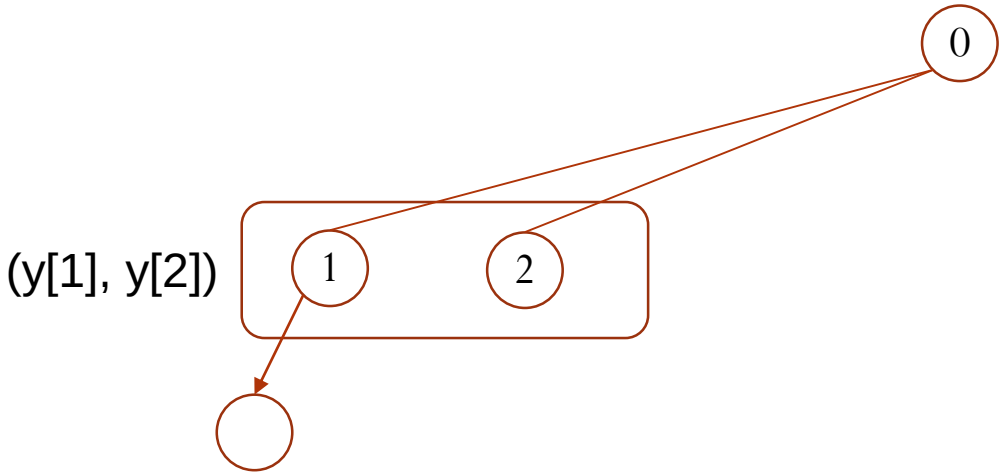
Bài toán CVRP

- K = 2



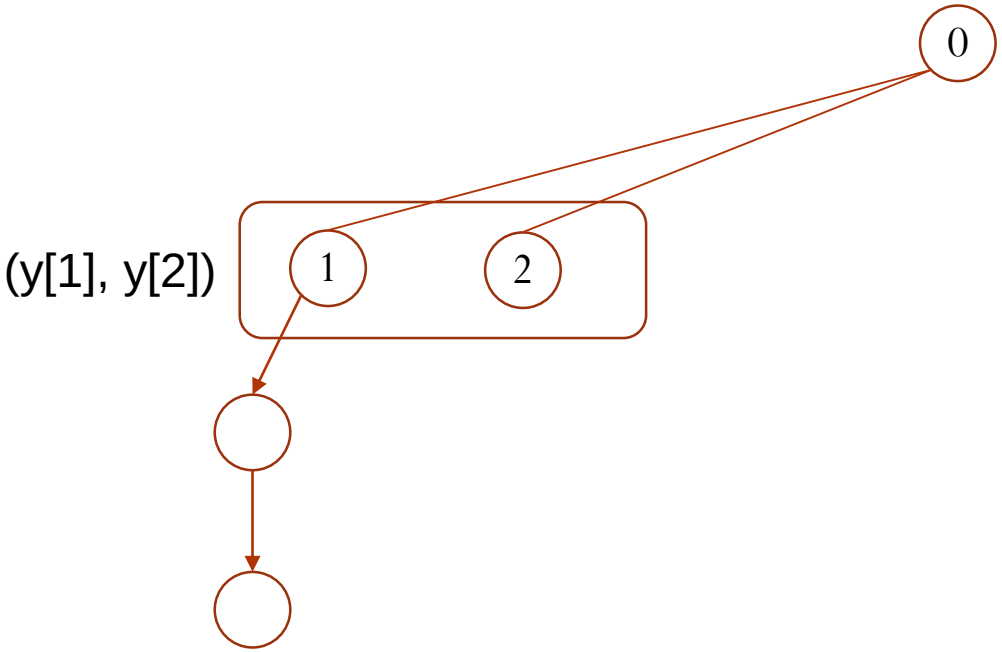
Bài toán CVRP

- K = 2



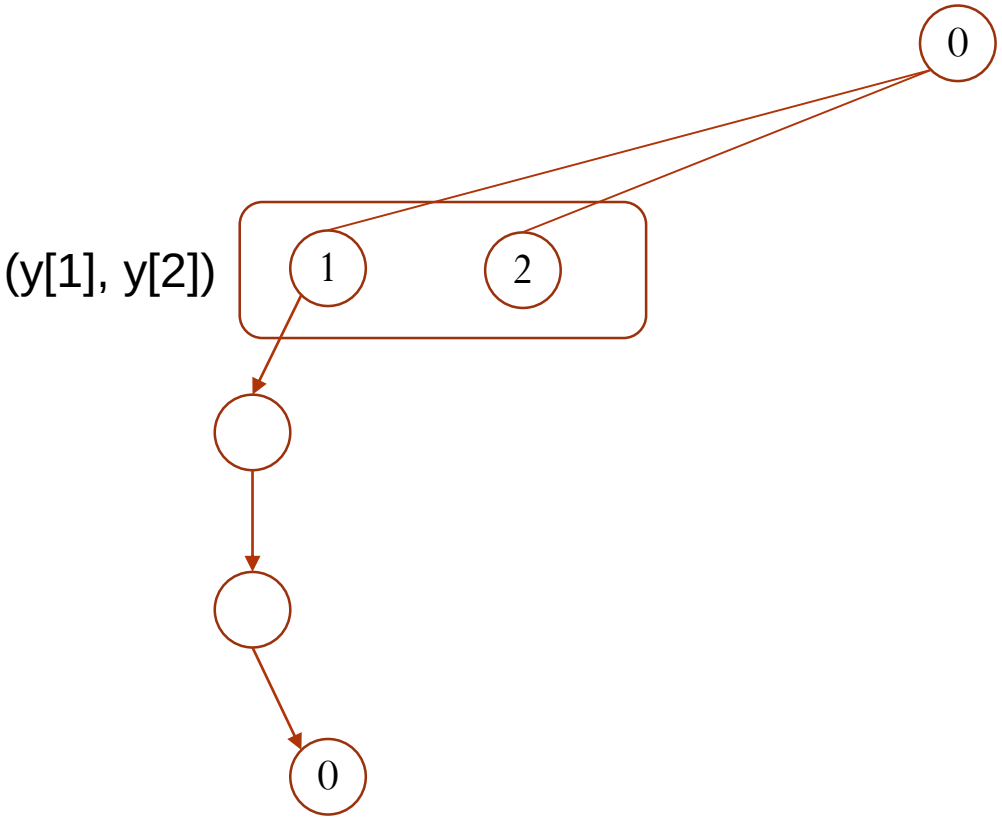
Bài toán CVRP

- K = 2



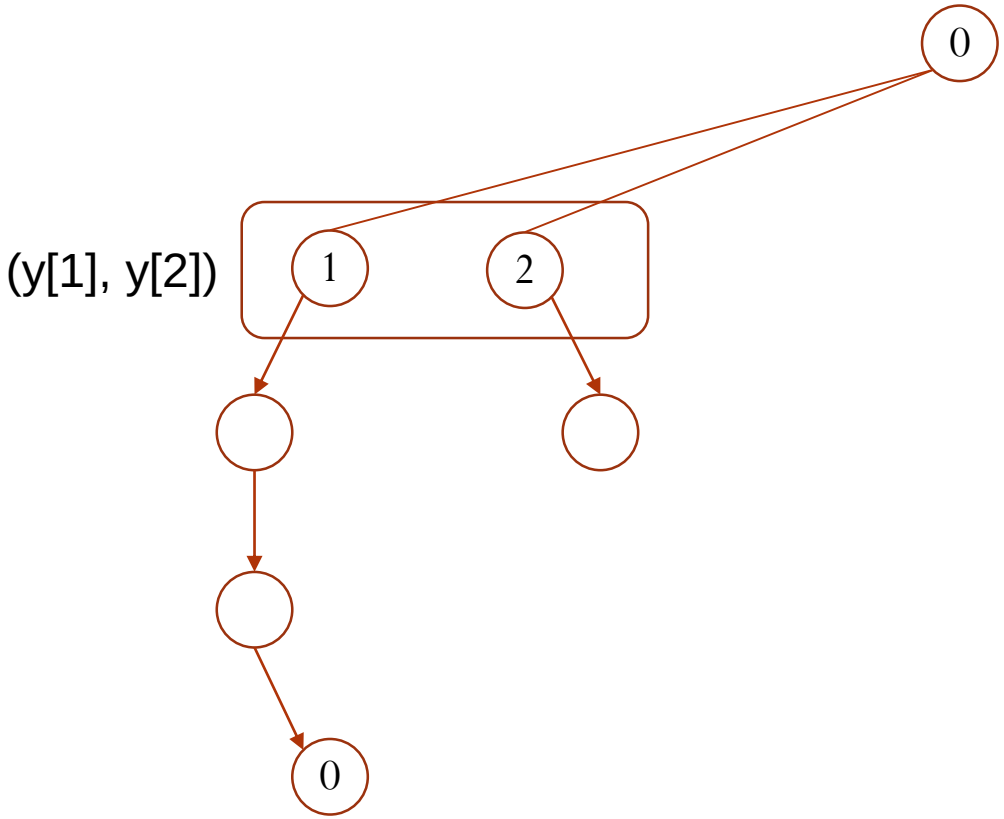
Bài toán CVRP

- K = 2



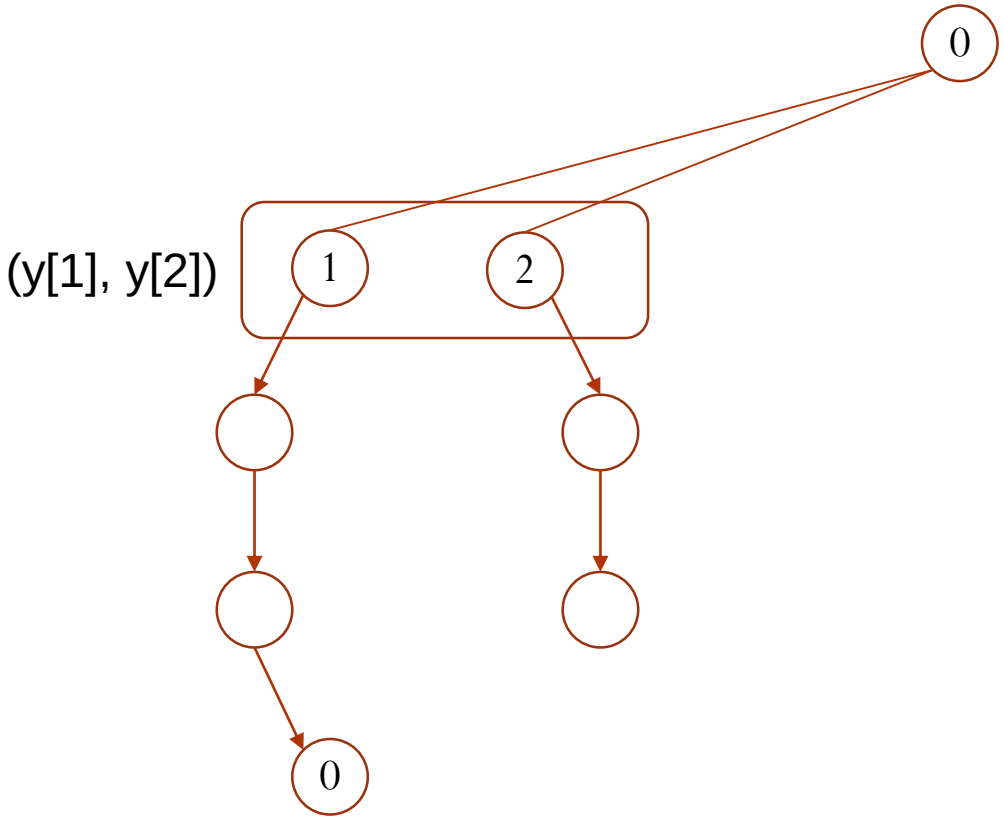
Bài toán CVRP

- K = 2



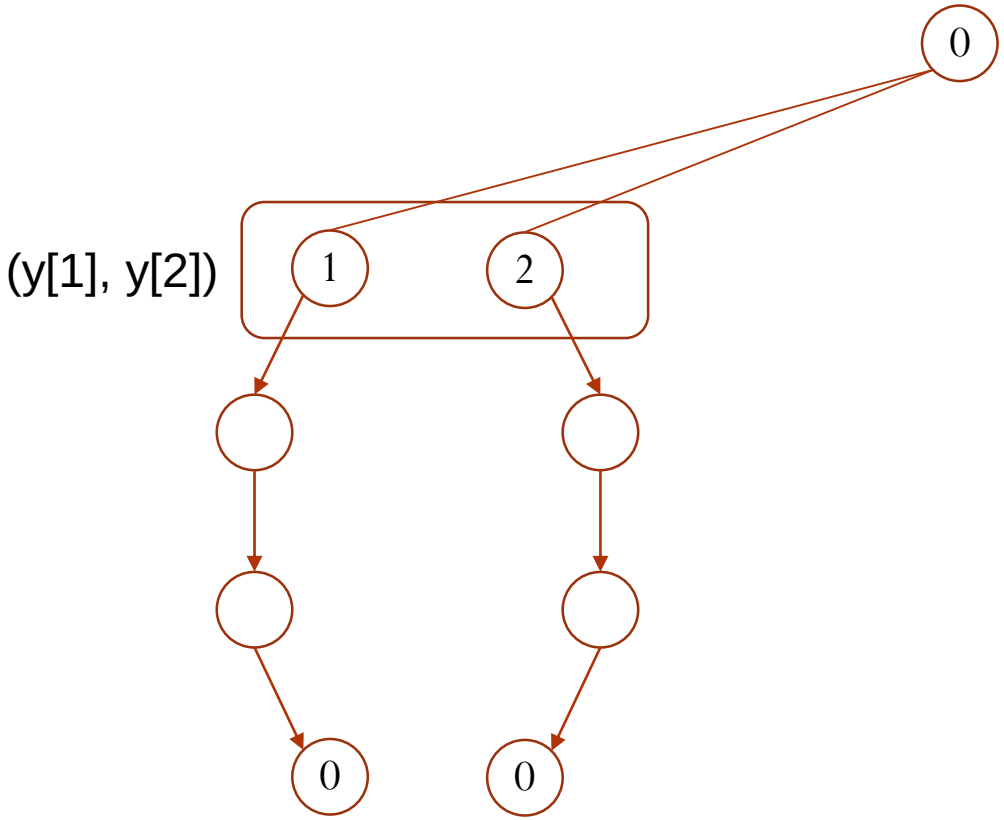
Bài toán CVRP

- K = 2



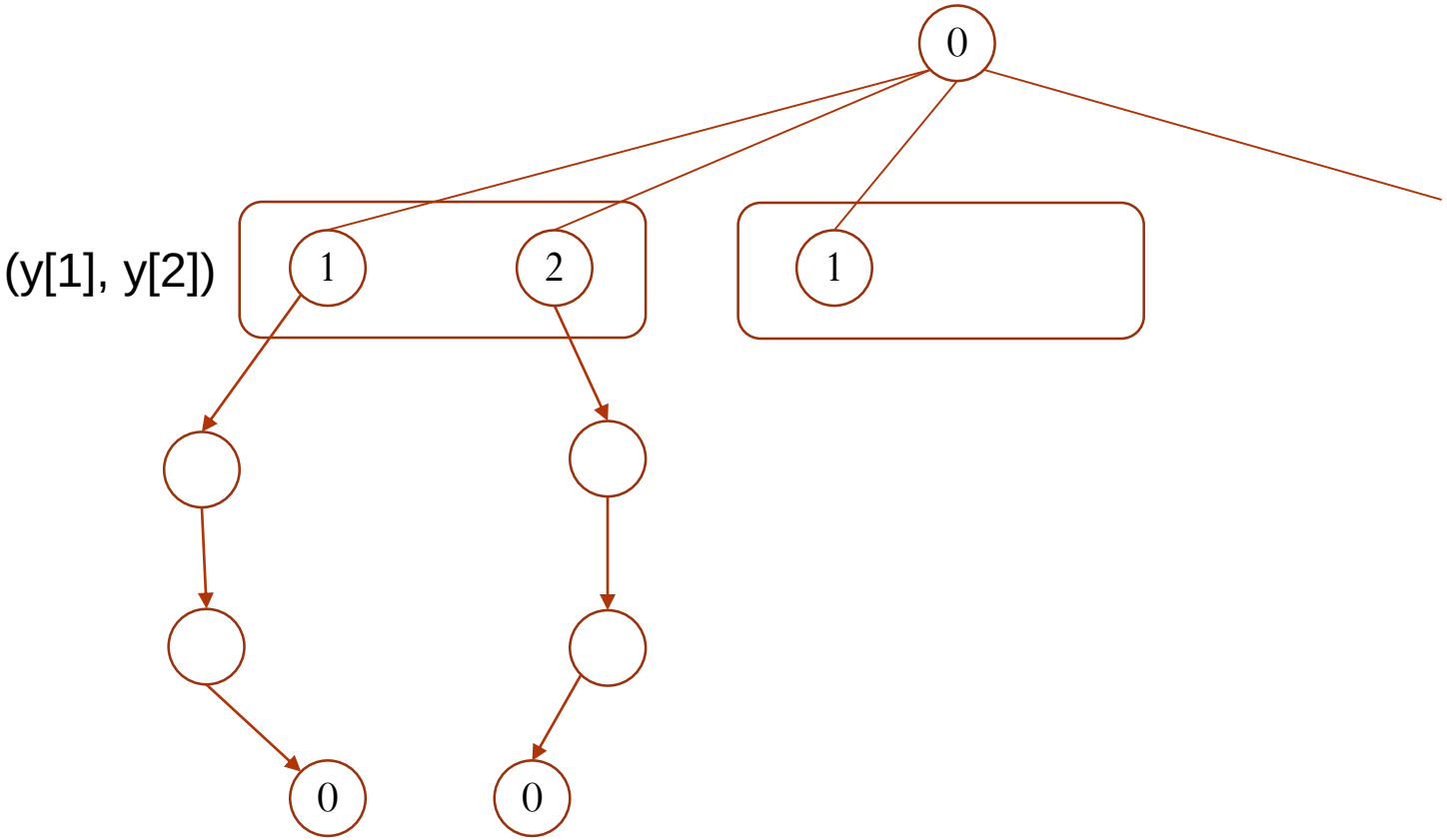
Bài toán CVRP

- K = 2



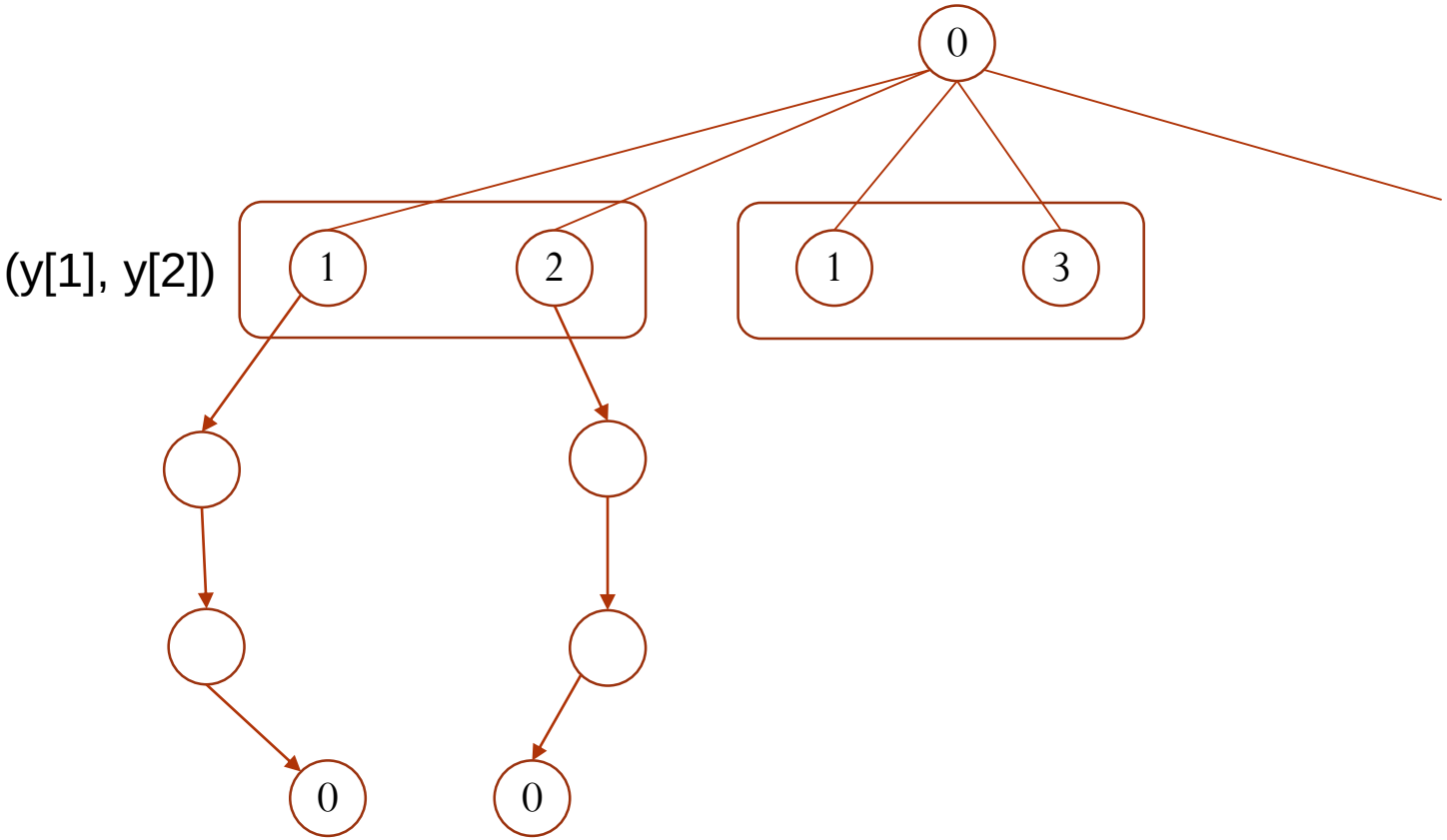
Bài toán CVRP

- K = 2



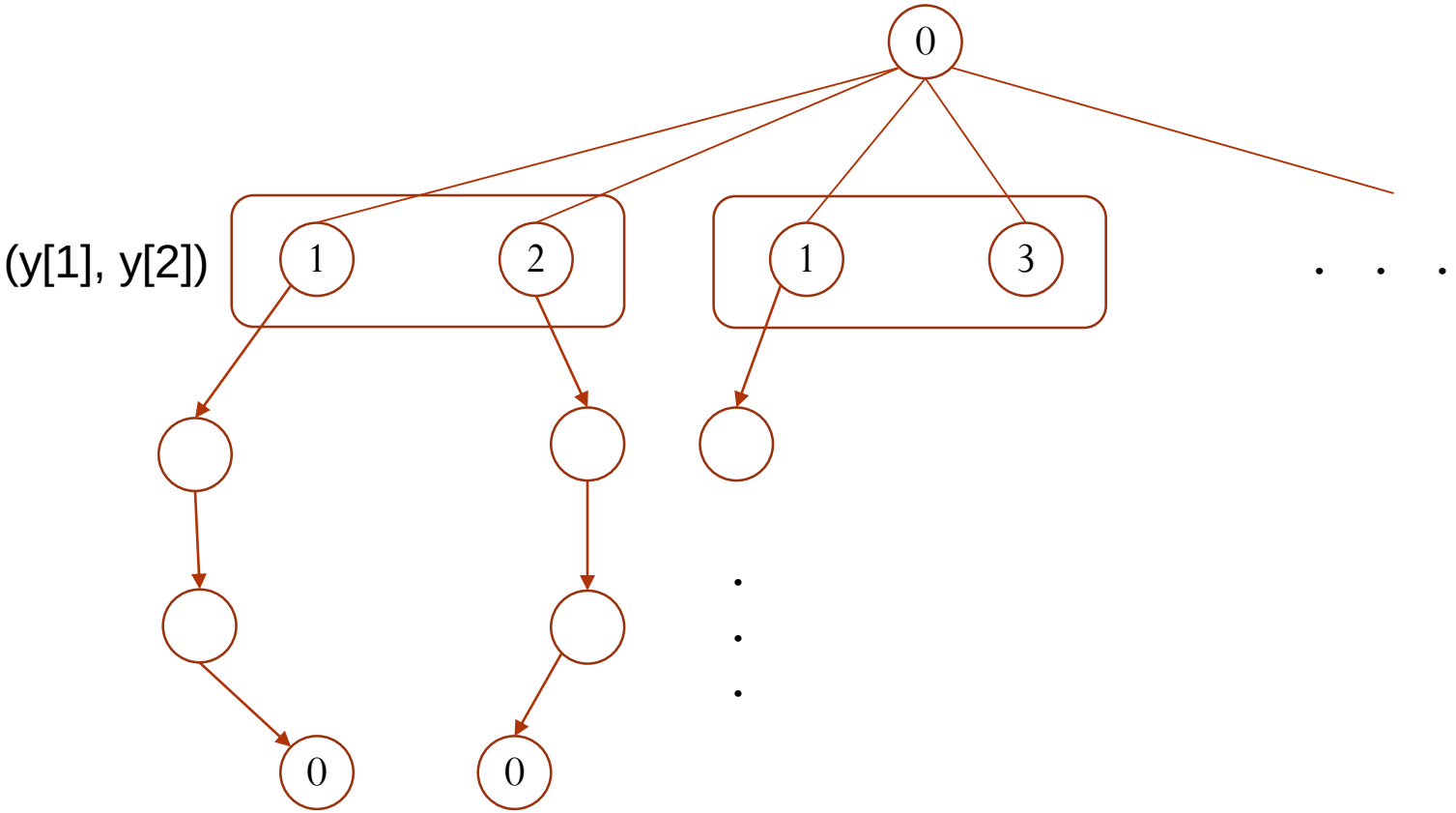
Bài toán CVRP

- K = 2



Bài toán CVRP

- K = 2



Bài toán CVRP

```
#include <stdio.h>
#define MAX 50

int n,K,Q;
int d[MAX];
int x[MAX]; // x[i] is the next point of i (i = 1,...,n), x[i] \in
            // {0,1,...,n}
int y[MAX];// y[k] is the start point of route k
int load[MAX];
int visited[MAX];// visited[i] = 1 means that client point i has been
visited
int segments;// number of segments accumulated
int nbRoutes;
int ans; // records number of solutions
```

Bài toán CVRP

```
void solution(){
    ans++;
    for(int k = 1; k <= K; k++){
        int s = y[k];
        printf("route[%d]:  0 ",k);
        for(int v = s; v != 0; v = x[v]){
            printf("%d ",v);
        }
        printf("0\n");
    }
    printf("--------------------------\n");
}
```

Bài toán CVRP

```
int checkX(int v,int k){
    if(v > 0 && visited[v]) return 0;
    if(load[k] + d[v] > Q) return 0;
    return 1;
}
void input(){
    scanf("%d%d%d",&n,&K,&Q);
    for(int i = 1; i <= n; i++){
        scanf("%d",&d[i]);
    }
    d[0] = 0;
}
```

Bài toán CVRP

```
void TRY_X(int s, int k){
    for(int v = 0; v <= n; v++){
        if(checkX(v,k)){
            x[s] = v;
            visited[v] = 1;  load[k] += d[v]; segments++;
            if(v > 0)   TRY_X(v,k);
            else{
                if(k == K){
                     if(segments == n+nbRoutes) solution();
                }else  TRY_X(y[k+1],k+1);
            }
            segments--;  load[k] -= d[v];   visited[v] = 0;
        }
    }
}
```

Bài toán CVRP

```
int checkY(int v, int k){
    if(v == 0) return 1;
    if(load[k] + d[v] > Q) return 0;
    return !visited[v];
}
```

Bài toán CVRP

```
void TRY_Y(int k){
    for(int v = y[k-1] + 1; v <= n; v++){// 0 < y[1] < y[2] < . . . . < y[K]
        if(checkY(v,k)){
            y[k] = v;
            segments += 1;
            visited[v] = 1;    load[k] += d[v];
            if(k < K){
                TRY_Y(k+1);
            }else{
                nbRoutes = segments;
                TRY_X(y[1],1);// du bo y[1],...,y[K], bat dau duyet cho diem tiep
                              // theo cua y[1]
            }
            load[k] -= d[v];  visited[v] = 0;
            segments -= 1;
        }
    }
}
```

Bài toán CVRP

```
void solve(){
    for(int v = 1; v <= n; v++) visited[v] = 0;
    y[0] = 0;
    ans = 0;
    TRY_Y(1);
    printf(“Number of solutions = %d",ans);
}
int main(){
    input();
    solve();
}
```

Thuật toán nhánh và cận

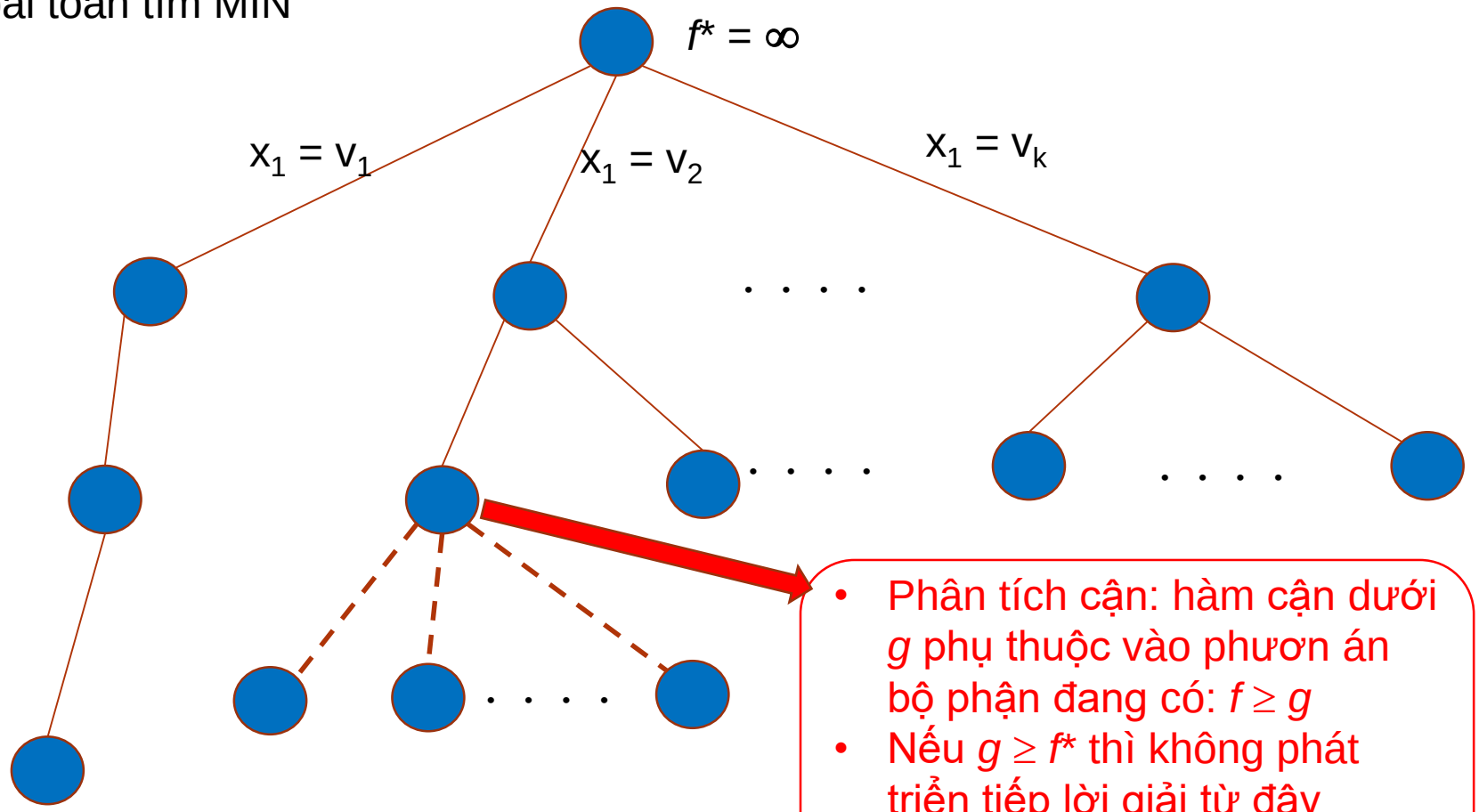
- Một trong số các phương pháp giải bài toán tối ưu tổ hợp
  - Dùng đệ quy quay lui để duyệt toàn bộ không gian lời giải
  - Dùng kỹ thuật phân tích đánh giá cận để cắt bớt nhánh tìm kiếm không có ích

Thuật toán nhánh và cận

- Bài toán tối ưu tổ hợp
  - Phương án $x = (x_1, x_2, \ldots, x_n)$ trong đó $x_i \in A_i$ cho trước
  - Phương án thoả mãn ràng buộc C
  - Hàm mục tiêu $f(x) \rightarrow$ min (max)

Thuật toán nhánh và cận

Xét bài toán tìm MIN



- Phân tích cận: hàm cận dưới *g* phụ thuộc vào phương án bộ phận đang có: $f \geq g$
- Nếu $g \geq f^*$ thì không phát triển tiếp lời giải từ đây

Thuật toán nhánh và cận

- Duyệt nhánh và cận:
  - Phương án bộ phận (a_1,…, a_k) trong đó a_1 gán cho x_1,… a_k gán cho x_k
  - Phương án (a_1,…, a_k, b_{k+1},…,b_n) là một phương án đầy đủ được phát triển từ (a_1,…,a_k) trong đó b_{k+1} gán cho x_{k+1},…, b_n được gán cho x_n
  - Với mỗi phương án bộ phận (x_1,…, x_k), hàm cận dưới $g(x_1,…, x_k)$ có giá trị không lớn hơn giá trị hàm mục tiêu của phương án đầy đủ phát triển từ (x_1,…,x_k)
  - Nếu $g(x_1,…,x_k) \geq f^*$ thì không phát triển lời giải từ (x_1,…,x_k)

```
TRY(k) {
  Foreach v thuộc A_k
      if check(v,k) {
        x_k = v;
        if(k = n) {
            ghi_nhan_cau_hinh;
            cập nhật kỷ lục f*;
         } {
            if g(x1,…,x_k) < f*
              TRY(k+1);
        }
       }
}
Main()
{ f* = ∞;
  TRY(1);
}
```

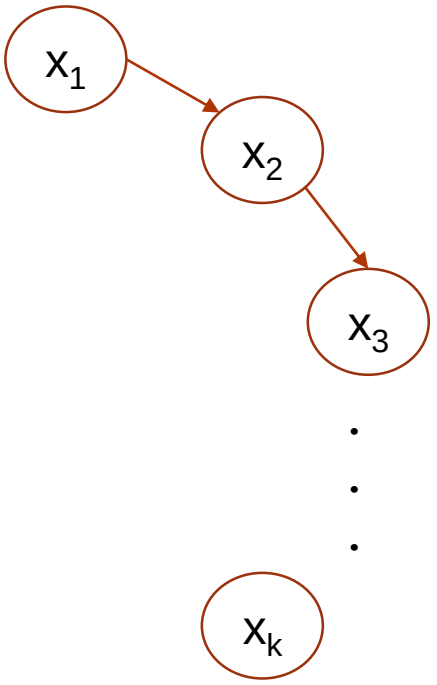
Thuật toán nhánh và cận

- Bài toán TSP
  - c_m là chi phí nhỏ nhất trong số các chi phí đi giữa 2 thành phố khác nhau
  - Phương án bộ phận $(x_1,\ldots, x_k)$
    - Chi phí bộ phận $\bar{f} = c(x_1, x_2) + c(x_2, x_3) + \ldots + c(x_{k-1}, x_k)$
    - Hàm cận dưới

      $$g(x_1,\ldots, x_k) = \bar{f} + c_m \times (n-k+1)$$

    - Hàm mục tiêu f của các lời giải pháp triển từ lời giải hiện tại

      $$f \geq g(x_1,\ldots, x_k)$$



Thuật toán nhánh và cận

```
void TRY(int k){
  for(int v = 1; v <= n; v++){
    if(marked[v] == false){
      x[k] = v;
      f = f + c[x[k-1]][x[k]];
      marked[v] = true;
      if(k == n){
        solution();
      }else{
        int g = f + cmin*(n-k+1);
        if(g < f_min)
          TRY(k+1);
      }
      marked[v] = false;
      f = f - c[x[k-1]][x[k]];
    }
  }
}
```

```
void solution() {
  if(f + c[x[n]][x[1]] < f_min){
    f_min = f + c[x[n]][x[1]];
  }
}
void main() {
  f_min = 9999999999;
  for(int v = 1; v <= n; v++)
    marked[v] = false;
  x[1] = 1; marked[1] = true;
  f = 0;
  TRY(2);
}
```