Training 3: Exhaustive Search

Ngoc Bui

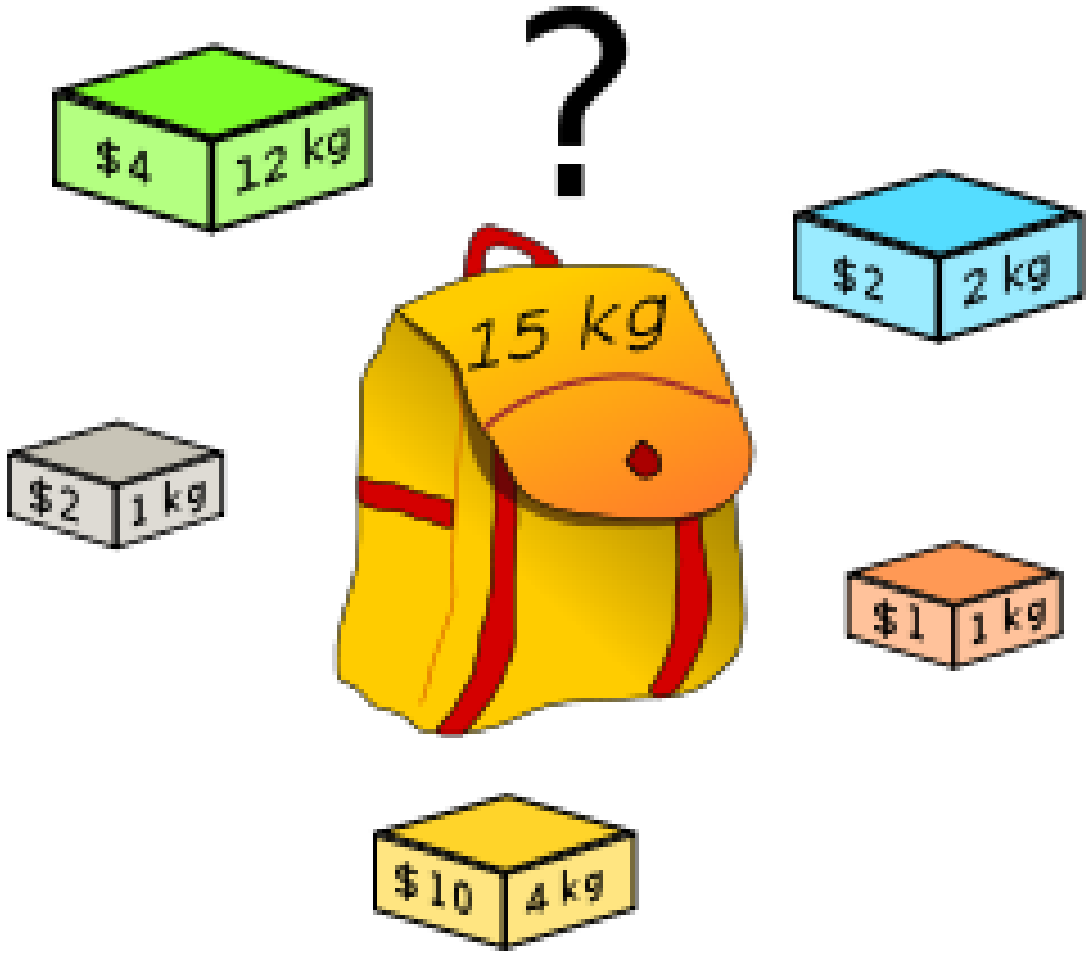
Hanoi University of Science and Technology

Hanoi, 2020

Outline

03. KNAPSAC (Thai9cdb)

- Cho một cái túi có thể chứa tối đa khối lượng M và n đồ vật. Mỗi đồ vật có khối lượng và giá trị của nó.
- Tìm cách lấy một số đồ vật sao cho tổng khối lượng không vượt quá M và tổng giá trị lớn nhất có thể.


?
$4
12 kg
$2
2 kg
15 kg
$2
1 kg
$1
1 kg
$10
4 kg

Thuật toán 1

- Mỗi cách chọn lấy các đồ vật tương ứng với một dãy nhị phân độ dài n. Bit thứ i là 0/1 tương ứng là không lấy/có lấy đồ vật thứ i.
- Xét hết các xâu nhị phân độ dài n và tìm nghiệm tốt nhất.
- Để xét hết các xâu nhị phân độ dài n, có thể dùng đệ quy - quay lui hoặc chuyển đổi giữa thập phân với nhị phân.
- Độ phức tạp $O(2^n \times n)$

Code 1a

```
void duyet(int i,int sum,int val){
    if (i>n){
        if (val>best) best=val;
        return;
    }
    duyet(i+1,sum,val);//bit 0
    if (sum+m[i]<=M)
        duyet(i+1,sum+m[i],val+v[i]);//bit 1
}
```

Code 1b

```
main(){
    cin >> n >> M;
    for (int i = 0; i < n; ++i)
    cin >> m[i] >> v[i];
    int ans = 0;
    for (int mask = 1 << n; mask--; ){//mask = 0..2^n
        int sumM = 0, sumV = 0;
        for (int i = 0; i < n; ++i)
            if (mask >> i & 1){//bit thu i = 1
                sumM += m[i];
                sumV += v[i];
            }
        if (sumM <= M) ans = max(ans, sumV);
    }
    cout << ans;
}
```

Thuật toán 2

- Chia tập đồ vật làm hai phần A và B. Mỗi cách chọn lấy các đồ vật tương ứng với một cách lấy bên A kết hợp với một cách lấy bên B.
- Ý tưởng chính ở đây là lưu trữ hết các cách lấy bên B và sắp xếp trước theo một thứ tự. Sau đó với mỗi cách lấy bên A, ta có thể tìm kiếm nghiệm tối ưu bên B một cách nhanh chóng.
- Giả sử cách lấy tối ưu là lấy m_A bên A và m_B bên B, ta sẽ xét tuần tự từng m_A một và tìm kiếm nhị phân m_B.
- Độ phức tạp $O(2^A \times \log(2^B) + 2^B) = O(2^{n/2} \times n)$ nếu chọn $|A| = |B| = n / 2$

Code 2

//S1, S2 là tập các cách lấy bên A, bên B. maxV[j] là max(S2[0..j].v) //S1 và S2 được tính bằng đệ quy - quay lui

```
int ans = -1e9;
for(int i=0;i<S1.size();++i){
    int L = 0, H = S2.size()-1, j = -1;
    while (L<=H){
        int m = (L+H)/2;
        if (S1[i].m + S2[m].m <= M){
            j = m;
            L = m+1;
        } else H = m-1;
    }
    if (j!=-1){
        ans = max(ans, S1[i].v + maxV[j]);
    }
}
cout << ans;
```

03. TSP (Thai9cdb)

- Cho một đồ thị đầy đủ có trọng số.
- Tìm một cách di chuyển qua mỗi đỉnh đúng một lần và quay về đỉnh xuất phát sao cho tổng trọng số các cạnh đi qua là nhỏ nhất (chu trình hamilton nhỏ nhất).

Thuật toán 1

- Mỗi cách di chuyển ứng với một hoán vị của n đỉnh.
- Xét hết các hoán vị và tìm nghiệm tốt nhất.
- Để xét hết các hoán vị, có thể dùng đệ quy - quay lui hoặc thuật toán sinh kế tiếp.

Code 1

//i là số đỉnh đã đi qua, sum là tổng trọng số đường đi. Mảng x[.] toàn cục lưu danh sách các đỉnh đi qua theo thứ tự. Mảng c[.][.] toàn cục lưu ma trận trọng số

```
void duyet(int i,int sum){
    if (i==n+1){
        if (sum+c[x[n]][1] < best)
            best=sum+c[x[n]][1];
        return;
    }
    for (int j=2;j<=n;++j)
        if (!mark[j]){
            if (sum+c[x[i-1]][j]<best){
                mark[j]=1;
                x[i]=j;
                duyet(i+1,sum+c[x[i-1]][j]);
                mark[j]=0;
            }
        }
}
```

Thuật toán 2

- Để ý cây đệ quy (tạo ra từ quá trình duyệt) có rất nhiều cây con giống nhau. Ta có thể chỉ duyệt một lần và lưu trữ lại kết quả.
- Trạng thái duyệt là danh sách các đỉnh đã đi qua và đỉnh hiện tại đang đứng (hay danh sách các số đã xuất hiện và số cuối cùng trong phần hoán vị đã xây dựng). Hai cây con giống nhau nếu trạng thái tại nút gốc của chúng giống nhau.
- Ta có thể lưu trữ nghiệm tối ưu cho từng trạng thái để không phải duyệt lại nhiều lần. Sử dụng kỹ thuật bitmask để lưu trữ thuận tiện hơn.

Code 2

//p là đỉnh đang đứng, X là tập các đỉnh đã đi qua

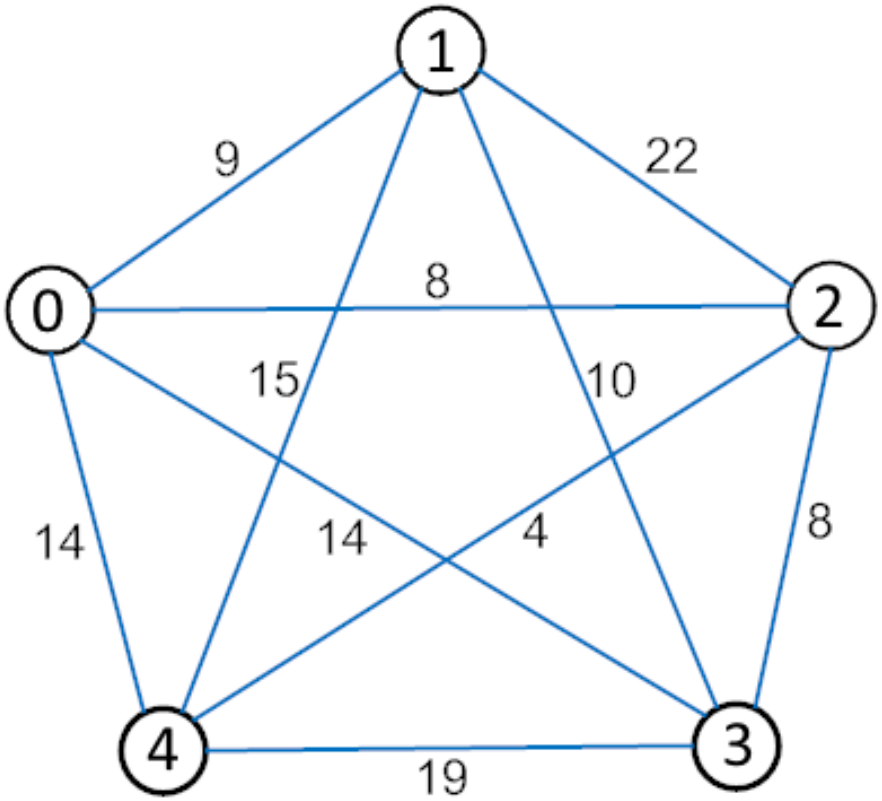
```
int duyet(int X, int p){
    if (__builtin_popcount(X) == n) return c[p][0];
    if (save[X][p] != -1) return save[X][p];
    int ans = 2e9;
    for (int s = 0; s < n; ++s)
        if ((X >> s & 1) == 0){
            ans = min(ans, c[p][s]
            + duyet(1 << s | X, s));
        }
    save[X][p] = ans;
    return ans;
}
```

03.TAXI(TungTT)

- Nêu ra lần đầu tiên năm 1930 về tối ưu hóa
- Dưới dạng bài toán “The Saleman Problem”

Phát biểu bài toán gốc

- Một tài xế Taxi xuất phát từ điểm 0, và nếu khoảng cách giữa hai điểm bất kỳ được biết thì đâu là đường đi ngắn nhất mà người Taxi có thể thực hiện sao cho đi hết tất cả các điểm mỗi điểm một lần để quay về lại điểm A.
- Đầu vào: khoảng cách giữa các điểm, tài xế Taxi xuất phát từ điểm 0, và có n điểm từ $1, 2, 3, ..n$ cần đi qua.
- Đầu ra: đường đi ngắn nhất $0 \rightarrow i \rightarrow j \ldots \rightarrow 0$
- Dưới dạng đồ thị: bài toán người lái taxi được mô hình hóa như một đồ thị vô hướng có trọng số, trong đó mỗi điểm đến là một đỉnh của đồ thị, đường đi từ một điểm đến điểm khác là khoảng cách hay chính là độ dài cạnh.



- Tổng quãng đường đi từ
  $0 \to 2 \to 4 \to 1 \to 3 \to 0 = 8 + 4 + 15 + 10 + 4 = 51$
- Tổng quãng đường đi từ
  $0 \to 1 \to 4 \to 2 \to 3 \to 0 = 9 + 15 + 4 + 8 + 4 = 40$

Ứng dụng

- Lập kế hoạch như bài toán Taxi là tối ưu trong quãng đường phục vụ, bài toán Người bán hàng,...
- Thiết kế vi mạch $\rightarrow$ Tối ưu về đường nối, điểm hàn
- Trong lĩnh vực phân tích gen sinh học
- Trong lĩnh vực du lịch

TAXI

- Có n hành khách được đánh số từ 1 tới n.
- Có $2n + 1$ địa điểm được đánh số từ 0 tới $2n$
- Hành khách thứ i muốn đi từ địa điểm thứ i đến địa điểm thứ $i + n$
- Taxi xuất phát ở địa điểm thứ 0 và phải phục vụ n hành khách và quay lại địa điểm thứ 0 sao cho không điểm nào được đi lại 2 lần và tại một thời điểm chỉ có 1 hành khách được phục vụ.
- Cho khoảng cách giữa các địa điểm. Tính quãng đường nhỏ nhất mà tài xế Taxi phải đi.

Thuật toán

- Nhận xét với mỗi cách chọn đường đi ta sẽ ánh xạ về được một hoán vị từ 1 đến n.
- Ta duyệt hết $n!$ hoán vị.
- Với mỗi hoán vị tính toán khoảng cách phải di chuyển.
- Độ phức tạp thuật toán $O(n! * n)$, có thể cải tiến xuống $O(n!)$ hoặc thậm chí $O(2^n * n^2)$.

Công thức

- Gọi $c[i][j]$ là khoảng cách di chuyển từ địa điểm i đến địa điểm j
- Gọi một hoán vị có dạng $a_1, a_2, ..., a_n$
- Công thức :
  $\sum_{i=1}^{n} c[a_i][a_i + n] + \sum_{i=1}^{n-1} c[a_i + n][a_{i+1}] + c[0][a_1] + c[a_n + n][0]$

Code

```
int main() {
  for (int i = 1; i <= n; i++) x[i] = i;
    int ans = INF;
    do {
      int cost = 0;
      for (int i = 1; i <= n; i++)
          cost += c[x[i]][x[i]+n] +
                  (i!=n)*c[x[i]+n][x[i+1]];
      cost += c[0][x[1]] + c[x[n]+n][0];
      ans = min(ans, cost);
    } while ( next_permutation(x+1,x+n+1) );
    cout << ans;
}
```

03. CBUS (ngocbh)

- Có n hành khách $1, 2, \ldots, n$, hành khách i cần di chuyển từ địa điểm i đến địa điểm $i + n$
- Xe khách xuất phát ở địa điểm 0 và có thể chứa tối đa k hành khách
- Cho ma trận c với $c(i, j)$ là khoảng cách di chuyển từ địa điểm i đến địa điểm j
- Tính khoảng cách ngắn nhất để xe khách phục vụ hết n hành khách và quay trở về địa điểm 0
- **Lưu ý:** Ngoại trừ địa điểm 0, các địa điểm khác chỉ được thăm tối đa 1 lần

Thuật toán - Duyệt toàn bộ

- Sử dụng mảng trạng thái x_i đánh dấu trạng thái của các khách i.
  - $x_i = 0$ khách i chưa lên xe.
  - $x_i = 1$ khách i đã lên xe.
  - $x_i = 2$ khách i đã xuống xe.
- Mảng $x_1, x_2, ...x_n$ khởi tạo là $0, 0, ...0$
- Trạng thái mục tiêu: khi đã trả toàn bộ khách $2, 2, ...2$
- Duyệt đệ quy tất cả các đường đi từ trạng thái $0, 0, ...0$ đến trạng thái $2, 2, ...2$
- Đưa ra đường đi có chi phí nhỏ nhất và đảm bảo điều kiện số bit 1 trong mỗi trạng thái đi qua không quá k.

Thuật toán - Duyệt toàn bộ

- Sử dụng kỹ thuật nhánh cận để giảm số lần gọi đệ quy. Xe khách luôn cần đi qua $2 \times N + 1$ cạnh. Gọi f là độ dài đường đi hiện tại, r là số cạnh mà xe khách còn phải đi qua, c_{min} là độ dài cạnh nhỏ nhất, ta có công thức cận:

$$bound = f + r \times c_{min} \tag{1}$$

Code

```
void search(int u, int fs)
{
  if ( completed == 2*n ) {
    ans = min(ans, scost + c[u][0]);
  } else {
    for (int v = 1; v <= n; v++) {
      //calc bound
      int bound = scost + c_min * (2*n - completed - 1);
      if ( x[v] == 0 && fs > 0 && bound + c[u][v] < ans) {
        // update status of passenger v from 0 -> 1
        x[v] = 1;
        // update cost
        scost += c[u][v];
        // update number of completed mask
        completed += 1;
        // recursive
        search(v, fs-1);
        // trace back data before recursive
        x[v] = 0;
        scost -= c[u][v];
        completed -= 1;
      } else if ( x[v] == 1 && bound + c[u][v+n] < ans ) {
        // update status of passenger v from 1 -> 2
        // same behavior
        x[v] = 2;
        scost += c[u][v+n];
        completed += 1;
        search(v+n,fs+1);
        x[v] = 1;
        scost -= c[u][v+n];
        completed -= 1;
      }
```

Thuật toán - Quy hoạch động trạng thái

- Nhận xét: các trạng thái bị duyệt đi duyệt lại rất nhiều.
- $\rightarrow$ sử dụng quy hoạch động trạng thái lưu lại kết quả.

Code

```
int get_bit(int mask,int i) {
    for (int j = i; j < n; j++)
        mask /= 3;
    return mask % 3;
}

int set_bit(int mask, int i, int v) {
    int x = get_bit(mask, i);
    mask -= x*p3[n-i];
    mask += v*p3[n-i];
    return mask;
}

int is_valid(int mask,int k)
{
    while ( mask > 0 ) {
        k -= ((mask % 3) == 1);
        mask /= 3;
    }
    if ( k >= 0 ) return 1;
    else return 0;
}

for (int mask = 0; mask < p3[n]; mask++) {
  valid[mask] = is_valid(mask,k);
  for (int i = 0; i <= n; i++)
    dp[mask][i] = INF;
}
```

Code

```
for (int i = 1; i <= n; i++)
dp[set_bit(0,i,1)][i] = c[0][i];

for (int mask = 1; mask < p3[n]; mask++) {
  if (!valid[mask]) continue;
  for (int i = 1; i <= n; i++) {
    int x = get_bit(mask,i);
    if ( x == 0 ) continue;
    for (int j = 0; j <= n; j++) {
      int pre_mask = set_bit(mask,i,x-1);
      int y = get_bit(pre_mask,j);
      if (y == 0 or !valid[pre_mask]) continue;
      dp[mask][i] = min(dp[mask][i], dp[pre_mask][j] + c[j+(y-1)*n][i+(x-1)*n]);
    }
  }

}

int ans = INF;
for (int i = 1; i <= n; i++)
  ans = min(ans, dp[p3[n]-1][i] + c[i+n][0]);
```

03. CVRP (PQD)

- A fleet of K identical trucks having capacity Q need to be scheduled to deliver pepsi packages from a central depot 0 to clients $1, 2, \ldots, n$. Each client i requests $d[i]$ packages.
- Solution: For each truck, a route from depot, visiting clients and returning to the depot for deliverying requested pepsi packages such that:
  - Each client is visited exactly by one route
  - Total number of packages requested by clients of each truck cannot exceed its capacity
- Goal
  - Compute number R of solutions

03. CVRP (PQD)

- Input
  - Line 1: n, K, Q $(2 \leq n \leq 10, 1 \leq K \leq 5, 1 \leq Q \leq 20)$
  - Line 2: $d[1], ..., d[n]$ $(1 \leq d[i] \leq 10)$
- Output: $R \bmod 10^9 + 7$

03. CVRP

- Input
  - Có K xe tải, chở pepsi đến N khách hàng.
  - $\rightarrow$ Mỗi khách hàng có tối có thể nhận được pepsi từ 1 trong K xe tải
  - Có N^K trạng thái. Duyệt toàn bộ N^K trạng thái này, tính số trạng thái đảm bảo điều kiện số pepsi phải chở không quá Q

Code

```
void search(int i)
{
  if ( i > n ) {
    int res = 1;
    for (int j = 1; j <= k; j++)
      res = res * factorial[nt[j]];
    ans = ans + res;
    return;
  } else {
    for (int j = 1; j <= k; j++)
      if ( s[j] + a[i] <= q ) {
        x[i] = j;
        s[j] += a[i];
        nt[j] += 1;
        search(i+1);
        s[j] -= a[i];
        nt[j] -= 1;
      }
  }
}

factorial[0] = 1;
for (int i = 1; i <= n; i++)
  factorial[i] = factorial[i-1] * i;
factorial[0] = 0;

search(1);
cout << ans / factorial[k];
```

03. CVRP_OPT (ngocbh)

- Có K chiếc xe tải có cùng sức chứa Q thùng hàng cần chở hàng tới n khách hàng ở n điểm.
- Mỗi khách hàng cần chở $d[i]$ thùng hàng.
- Chi phí di chuyển từ điểm i đến điểm j là $c[i][j], 0 \leq i, j \leq n$.
- Mỗi xe tải cần chở hàng tới ít nhất một khách hàng và quay trở lại vị trí đầu 0.
- Mục tiêu: Tìm cách để định tuyến n xe sao cho tổng chi phí là nhỏ nhất.

Example

input

```
4 2 15
7 7 11 2
0 12 12 11 14
14 0 11 14 14
14 10 0 11 12
10 14 12 0 13
10 13 14 11 0
```

output

```
70
```

Các cách sắp xếp xe thỏa mãn đề bài:

- Xe 1: $1, 2$
- Xe 2: $3, 4$

Các hành trình thỏa mãn:

- Xe 1:
  - $0 \to 1 \to 2 \to 0$,
    $cost = 12 + 11 + 14 = 37$
  - $0 \to 2 \to 1 \to 0$,
    $cost = 12 + 10 + 14 = 36$
- Xe 2:
  - $0 \to 3 \to 4 \to 0$,
    $cost = 11 + 13 + 10 = 34$
  - $0 \to 4 \to 3 \to 0$,
    $cost = 14 + 11 + 10 = 35$

Example

input

```
4 2 15
7 7 11 2
0 12 12 11 14
14 0 11 14 14
14 10 0 11 12
10 14 12 0 13
10 13 14 11 0
```

output

```
70
```

Hành trình tối ưu

- Xe 1: $0 \to 2 \to 1 \to 0$,
  $cost = 12 + 10 + 14 = 36$
- Xe 2: $0 \to 3 \to 4 \to 0$,
  $cost = 11 + 13 + 10 = 34$

Tổng chi phí tối thiểu là 70.

Thuật toán

Từ các bước tính tay ta sẽ nhận thấy thuật toán:

- Bước 1: Chia phân tập cho n khách hàng, áp dụng tương tự bài CVRP_COUNT.
  - Xe 1: $1, 2$, Xe 2: $3, 4$
- Bước 2: Mỗi tập là danh sách các khách hàng mà xe i phải đến. tìm đường đi tối ưu cho mỗi xe $\rightarrow$ bài toán quy về bài toán TSP cơ bản.
  - Xe 1: $0 \rightarrow 2 \rightarrow 1 \rightarrow 0$, $cost = 12 + 10 + 14 = 36$
  - Xe 2: $0 \rightarrow 3 \rightarrow 4 \rightarrow 0$, $cost = 11 + 13 + 10 = 34$

Code

Hàm search phân tập tương tự CVRP_COUNT

```
void search(int i,int minj)
{
  if ( i > n ) {
    ans = min(ans,check_results());
  } else {
    for (int j = 1; j <= k; j++)
      if ( s[j] + a[i] <= q ) {
        x[i] = j;
        if (nt[j] == 1 ) {
          if (j > minj)
            search(i+1, j);
        } else
          search(i+1,minj);
      }
  }
}
```

Code

Kiểm tra chi phi của cách phân tập trên:

```
int check_results()
{
  int ret = 0;
  for (int i = 1; i <= k; i++) {
    vector<int> clients;
    for (int j = 1; j <= n; j++)
      if ( x[j] == i )
        clients.push_back(j);
    if (clients.size() == 0)
      return INF;
    ret += optimize_route(clients);
  }
  return ret;
}
```

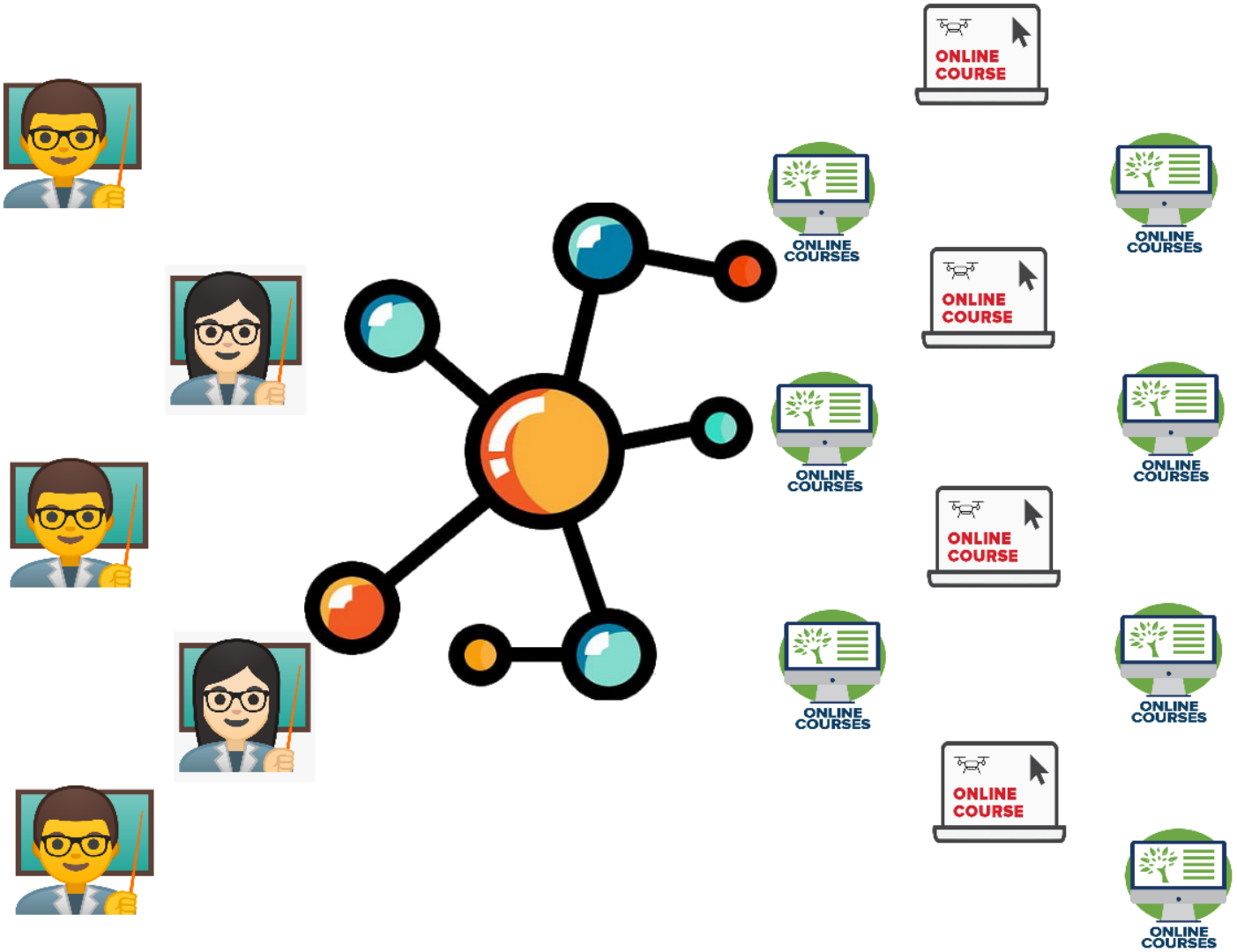
Code

Với mỗi tập, giải bài toán TSP tương đương:

```
int optimize_route(vector<int> clients)
{
  int ret = INF;
  do {
    int cost = 0;
    int m = clients.size();
    for (int i = 0; i < m-1; i++)
      cost += c[clients[i]][clients[i+1]];
    cost += c[0][clients[0]] + c[clients[m-1]][0];
    ret = min(ret, cost);
  } while ( next_permutation(clients.begin(), clients.en
  return ret;
}
```

03. BCA (ngocbh)

- Có n khóa học và m giáo viên, mỗi giáo viên có danh sách các khóa có thể dạy.
- Có danh sách các khóa học không thể để cùng một giáo viên dạy do trùng giờ.
- Load của một giáo viên là số khóa phải dạy của giáo viên đó.
- **Yêu cầu:** Tìm cách xếp lịch cho giáo viên sao cho Load tối đa của các giáo viên là tối thiểu.


ONLINE COURSE
ONLINE COURSES

Thuật toán I

- Sử dụng thuật toán vét cạn, duyệt toàn bộ khóa học, xếp giáo viên dạy khóa học đó.
- Sử dụng thuật toán nhánh cận:
  - Chọn khóa học chưa có người dạy có số giáo viên dạy ít nhất để phân công trước.
  - Nếu phân công cho giáo viên A môn X, mọi môn học trùng lịch với môn X không thể được dạy bởi giáo viên A sau này.
  - Nếu maxLoad hiện tại lớn hơn minLoad tối ưu thu được trước đó thì không duyệt nữa.

Code

```
void arrange(int i) {
    if (i > n) {
        // Check result...
    } else {
        int courseID = -1;
        int totalTeacher = 999;
        // Find non-assigned course the least
        // number of teachers to assign first.
        // ...

        if (courseID == -1) return;

        int maxLoad = 0;
        // Find current maxLoad...
        if (maxLoad >= minLoad) return;
```

Code

```
            sjList[courseID].selected = true;
            for (int j=1; j<=m; j++) {
                if (sjList[courseID].teacher[j]) {
                    bool tmp[31];
                    for (int k=1; k<=n; k++) {
                        tmp[k] = sjList[k].teacher[j];
                        if (conflict[courseID][k]) {
                            sjList[k].teacher[j]=false;
                        }
                    }
                    schedule[j][courseID] = true;
                    arrange(i+1);
                    schedule[j][courseID] = false;
                    for (int k=1; k<=n; k++)
                        sjList[k].teacher[j]=tmp[k];
                }
            }
            sjList[courseID].selected = false;
        }
```