ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN

CẤU TRÚC RỜI RẠC (CO1007)

Giải thuật tìm đường đi ngắn nhất
(Shortest path algorithms)

TP. HỒ CHÍ MINH, THÁNG 06/2025

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN

CẤU TRÚC RỜI RẠC (CO1007)

Instructor(s): Nguyễn Văn Minh Mẫn, *Mahidol University*
Nguyễn An Khương, *CSE-HCMUT*
Lê Hồng Trang, *CSE-HCMUT*
Trần Tuấn Anh, *CSE-HCMUT*
Trần Hồng Tài, *CSE-HCMUT*
Mai Xuân Toàn, *CSE-HCMUT*

Sinh viên thực hiện:**Lê Minh Hưng**
MSSV:**2252276**
Lớp:**CN01**

TP. HỒ CHÍ MINH, THÁNG 06/2025

MỤC LỤC

1 Giới thiệu bài toán TSP

Traveling Salesman Problem (TSP) là một bài toán tối ưu tổ hợp kinh điển: Cho một tập n thành phố và ma trận trọng số $w(u, v)$ giữa mỗi cặp (u, v). Tìm một chu trình Hamilton (đi qua mỗi thành phố đúng một lần rồi quay về điểm xuất phát) sao cho tổng trọng số chu trình là nhỏ nhất. TSP thuộc lớp NP-hard, nên với $n \leq 20$ ta có thể dùng `Bitmask-DP` exact, còn với $n > 20$ thường phải dùng heuristic hoặc cắt nhánh (`Branch-and-Bound`).

2 Mô tả bài toán

Đầu vào

- `edges[][3]`: mảng cạnh, mỗi hàng chứa `[u, v, w]` (ký tự `u`, `v` được mã hóa dưới dạng `int`, `w` là trọng số).
- `eCnt`: số cạnh.
- `start`: ký tự đỉnh xuất phát.

Đầu ra

- Chuỗi `"A B C ... A"` biểu diễn chu trình tối ưu đi qua tất cả đỉnh và quay về `start`.

3 Phương pháp giải: Bitmask-DP (Top-down Memoization)

3.1 Ý tưởng chính

Sử dụng một số nguyên `mask` $(0 \dots 2^n - 1)$ để đánh dấu tập đỉnh đã thăm: bit thứ `i` $= 1$ nếu đỉnh `i` đã ghé.

Định nghĩa hàm đệ quy:

$$f(\texttt{mask}, \texttt{last}) = \min \{ f(\texttt{mask} \cup \{\texttt{to}\}, \texttt{to}) + w(\texttt{last} \to \texttt{to}) \}$$

với base-case khi `mask` chứa tất cả đỉnh, ta cộng thêm chi phí quay về đỉnh khởi đầu.

3.2 State và chuyển tiếp

State: `dfs(mask, last)` – `mask`: bitmask các đỉnh đã thăm – `last`: đỉnh hiện tại.
Trả về chi phí tối thiểu để đi qua `mask` và dừng tại `last`, rồi kết thúc đóng chu trình về `startId`.

Công thức

```
dfs(mask, last):
    if mask == ALL:
        return cost_min(last -> startId)
    if memo[mask][last] != -1:
        return memo[mask][last]
    best = +inf
    for mỗi cạnh (last -> to):
        if (mask & (1<<to)) == 0:
            sub = dfs(mask | (1<<to), to)
            if sub < +inf:
                best = min(best, sub + w(last->to))
                pick[mask][last] = to
    memo[mask][last] = best
    return best
```

4 Cài đặt chi tiết

4.1 Chuẩn bị dữ liệu

1.Ánh xạ ký tự → chỉ số (0...n-1)

```
bool present[256]={0};
present[(unsigned char)start]=true;
for(int i=0;i<eCnt;i++){
  present[(unsigned char)edges[i][0]] = true;
  present[(unsigned char)edges[i][1]] = true;
}
int mapC[256]; fill_n(mapC,256,-1);
vector<char> rev;
for(int c=0;c<256;c++){
  if(present[c]){
    mapC[c]=rev.size();
    rev.push_back((char)c);
  }
```

```
}
int n = rev.size(), startId = mapC[(unsigned char)start];
```

2. Xây dựng đồ thị dạng adjacency-list

```
vector<vector<pair<int,int>>> graph(n);
for(int i=0;i<eCnt;i++){
    int u=mapC[(unsigned char)edges[i][0]],
        v=mapC[(unsigned char)edges[i][1]],
        w=edges[i][2];
    graph[u].push_back({v,w});
}

// Gom các cạnh trùng giữ trọng số nhỏ nhất
for(int u=0;u<n;u++){
    for(int i=0;i<graph[u].size();i++){
        for(int j=i+1;j<graph[u].size();j++){
            if(graph[u][i].first==graph[u][j].first){
                graph[u][i].second = min(
                    graph[u][i].second, graph[u][j].second
                );
                graph[u].erase(graph[u].begin()+j--);
            }
        }
    }
}
```

4.2 Đệ quy và Memoization

```
int ALL = (1<<n) - 1;
vector<vector<int>> memo(1<<n, vector<int>(n,-1)),
                    pick(1<<n, vector<int>(n,-1));

function<int(int,int)> dfs = [&](int mask, int last)->int {
    if(mask==ALL){
        int ret=INF;
        for(auto &e: graph[last])
            if(e.first==startId)
                ret = min(ret, e.second);
        return ret;
    }
    int &res = memo[mask][last];
    if(res!=-1) return res;

    int best=INF;
    for(auto &e: graph[last]){
        int to=e.first, w=e.second;
        if(mask & (1<<to)) continue;
        int sub = dfs(mask|(1<<to), to);
        if(sub<INF && sub+w<best){
            best = sub+w;
            pick[mask][last] = to;
        }
    }
    return res = best;
};

int minCost = dfs(1<<startId, startId);
if(minCost>=INF) return "";  // không có chu trình Hamilton
```

4.3 Backtracking dựng chu trình

```
vector<int> order;
int mask = 1<<startId, cur = startId;
while(true){
    order.push_back(cur);
    int nxt = pick[mask][cur];
    if(nxt<0) break;
    mask |= 1<<nxt;
    cur = nxt;
}
order.push_back(startId);  // đóng chu trình

// Chuyển thành chuỗi ký tự
string out;
for(int i=0;i<order.size();i++){
    if(i) out.push_back(' ');
    out.push_back(rev[order[i]]);
}
return out;
```

5 Phân tích độ phức tạp

Thời gian: mỗi state `(mask, last)` được tính một lần, tổng số state $= 2^n \times n$. Trong mỗi state ta duyệt các cạnh kề, nên tổng $O(E \cdot 2^n)$ hoặc $O(n^2 \cdot 2^n)$ với đồ thị đầy.

Bộ nhớ: $O(2^n \cdot n)$ cho hai bảng `memo` và `pick`.

Giới hạn: với $n \lesssim 20$, thuật toán chạy tốt trong vài giây. Với $n > 20$, cần xem xét heuristic (2-opt, Christofides) hoặc branch-and-bound.

6 Ưu nhược điểm và Mở rộng

Ưu điểm	Nhược điểm
Cho lời giải chính xác (exact).	Tốn nhiều bộ nhớ và thời gian khi n tăng.
Dễ cài đặt dưới dạng `DFS+memo`.	Giới hạn thực thi với $n \leq 20$.
Có thể mở rộng thêm table `next` để	Không áp dụng cho $n > 25$ nếu không cắt nhánh hoặc heuristic.

Mở rộng

1. **Branch-and-Bound / IDA**: dùng lower-bound MST để prune mạnh, mở rộng $n \approx 25$–30.
2. **Christofides (1.5-approx)**: cho đồ thị metric, độ phức tạp $O(n^3)$.
3. **Heuristic 2-opt, 3-opt**: cải thiện nhanh tour ban đầu, áp dụng với n rất lớn (1000+).

7 Kết luận

Dây là chi tiết cách giải TSP bằng **Bitmask-DP** (DFS có ghi nhớ), đảm bảo tìm chu trình tối ưu với độ phức tạp $\mathcal{O}(2^n \cdot n^2)$ và bộ nhớ $\mathcal{O}(2^n \cdot n)$. Phương pháp phù hợp với $n \lesssim 20$. Khi n lớn hơn, cần triển khai các kỹ thuật bổ trợ như *branch-and-bound* hoặc chuyển sang các thuật toán xấp xỉ.