Paradigmas de Resolução de Problemas

Programação Dinâmica: Problema do Caixeiro Viajante

Prof. Edson Alves – UnB/FGA

Sumário

Definição

Problema do Caixeiro Viajante

Dados N vértices $v_1, v_2, \ldots, v_N$ e os custos $w(v_i, v_j)$ das arestas, para todos $i \neq j$, o problema do caixeiro viajante (*travelling salesman problem* – TSP) é determinar uma travessia

$$t = s, v_{i_2}, v_{i_3}, \ldots, v_{i_N}, s$$

que tenha início em um vértice $v_{i_1} = s$ qualquer, passe por todos os demais uma única vez e retorne a $v_{i_{N+1}} = s$ com custo

$$C = \sum_{k=1}^{N} w(v_{i_k}, v_{i_{k+1}})$$

mínimo.

Características do problema do caixeiro viajante

- A interpretação mais comum é que os vértices representem cidades e o custo das arestas as distância entre elas
- Embora a definição apresentada exija que o grafo G seja completo, existem grafos que não são completos e que possuem solução
- Nestes grafos, caso não exista uma aresta entre v_i e v_j, basta fazer $w(v_i, v_j) = \infty$
- A travessia corresponde a uma permutação dos N vértices de G, sendo a única diferença a duplicação do primeiro vértice no final da travessia
- Na teoria dos grafos, tal travessia é denonimada um ciclo hamiltoniano, ou seja, um caminho cíclico que passa por todos os vértices uma única vez

Solução

Solução de programação dinâmica para o TSP

- Como cada ciclo hamiltoniano do grafo G corresponde a uma permutação de seus vértices, uma solução de busca completa para o TSP tem complexidade $O(N!)$, sendo aplicável para $N \approx 11$
- Pode-se observar que o TSP tem subproblemas repetidos: as permutações $v_1 v_2 v_3 \ldots v_N$ e $v_2 v_1 v_3 \ldots v_N$, por exemplo, compartilham um caminho composto por $N - 2$ vértices (a saber, $v_3 v_4 \ldots v_N$)
- Ele também tem subestrutura ótima, de modo que é um candidato natural a uma solução de programação dinâmica
- Uma importante observação: embora a definição não determine o vértice s, este vértice pode ser fixado arbitrariamente, uma vez que o caminho final é um ciclo

Solução de programação dinâmica para o TSP

- Assim, assuma que os vértices sejam numerados de 0 a $N-1$ e que $s=0$
- Seja $tsp(i, mask)$ o custo mínimo para passar por todos os vértices que não estão marcados em $mask$, partindo de i, e retonar ao vértice $s=0$
- O parâmetro $mask$ é um inteiro tal que o i-ésimo bit indica que o vértice i foi ou não visitado ($mask[i]=1$ e $mask[i]=0$, respectivamente)
- O caso base ocorre quando todos os vértices já foram visitados: resta, portanto, retornar ao vértice $s=0$
- Assim, $tsp(i, 2^N-1)=w(i,0)$

Solução de programação dinâmica para o TSP

- As transições consideram todos os vértices ainda não visitados
- Logo,
$$tsp(i, mask) = \min_j \{tsp(j, mask \mid mask[j] = 1) + w(i, j)\}$$
para todo j tal que $mask[j] = 0$
- Aqui, $mask \mid mask[j] = 1$ significa que o j-ésimo bit de $mask$ será ligado, mantendo-se os $bits$ que estavam ligados anteriormente
- A solução do problema será dada por $tsp(0, 1)$
- Há $O(N2^N)$ estados e as transições são feitas em $O(N)$, de modo que esta solução tem complexidade $O(N^2 2^N)$

Implementação *top-down* do TSP

```
int tsp(int i, int mask, int N)
{
    if (mask == (1 << N) - 1)
        return dist[i][0];

    if (st[i][mask] != -1)
        return st[i][mask];

    int res = oo;

    for (int j = 0; j < N; ++j)
    {
        if (mask & (1 << j))
            continue;

        res = min(res, tsp(j, mask | (1 << j), N) + dist[i][j]);
    }

    return (st[i][mask] = res);
}
```

Referências

1. **CORMEN**, Thomas H.; **LEISERSON**, Charles E.; **RIVEST**, Ronald; **STEIN**, Clifford. *Introduction to Algorithms*, 3rd Edition, MIT Press, 2009.
2. **LAARKSONEN**, Antti. *Competitive Programmer's Handbook*, 2017.
3. **HALIM**, Steve; **HALIM**, Felix. *Competitive Programming 3*, Lulu, 2013.