Estruturas Discretas - Segundo Trabalho

Prof. Marcus Vinicius S. Poggi de Aragão

Período de 2017.1

Gabriel Barbosa Diniz 1511211

Lucas Rodrigues 1510848

Mateus Ribeiro de Castro 1213068

29 de Maio de 2017

Observação$_1$: Os códigos fontes dos algoritmos referentes aos teoremas provados seguirá em anexo em um arquivo Jupyter Notebook para melhor entendimento, compilação, execução, testes, etc.

Observação$_2$: Os arquivos de entrada e saída (walk.in e walk.out) pedidos também estarão sendo enviados juntos, cumprindo as regras e exigências pedidas no trabalho.

1 Primeira Questão (Teorema 1)

Teorema 1 (i,j,q): Sabe-se determinar o prêmio máximo que o rei consegue coletar saindo da posição (i, j) e consumindo q unidades.

Vamos considerar um desarrolamento da matriz em 64 vértices distintos, com v_1 correspondente a $(1, 1)$, v_2 a $(1, 2)$, assim por diante. Os conceitos de vizinhança continuam valendo: v_1 tem como vizinhos $\{v_2, v_9, v_10\}$.

Considere, também, uma tabela cujas linhas correspondem aos vértices v, e as colunas ao custo q restante a ser utilizado. As células da tabela serão preenchidas com o prêmio máximo $P_{max}(v_{ij}, q)$, que se consegue a partir de um trajeto que inicie no vértice v_{ij} e que consuma q unidades.

Caso Base: Por indução em q, temos o caso base para $q = 0$, preencheremos a primeira coluna da tabela. Neste caso, não existem unidades para consumir, logo não poderemos sair da origem (i, j). Sendo assim, o prêmio máximo para ir até (i, j) será zero e para qualquer outro vértice será $-\infty$ (que representa a impossibilidade).

Hipótese Indutiva: Como hipótese indutiva, temos que o teorema é válido para $0 \leq q \leq Q$, portanto queremos provar que o teorema também é válido para $Q + 1$.

Passo Indutivo: Neste caso, para cada um dos vértices v, devemos encontrar o prêmio máximo que pode ser obtido chegando a v consumindo $Q+1$ unidades. Logo, podemos observar que, para que a condição acima seja satisfeita, no instante imediatamente anterior à chegada em v, estaríamos em um vértice v_n, vizinho de v, com $Q + 1 - q_v$ unidades consumidas, sendo q_v o custo associado ao vértice v. Visto que o prêmio p_v associado ao vértice v é constante, devemos escolher v_n de maneira que $P_{max}(v_n, Q+1-q_v)$ seja máximo, garantindo, assim, que $P_{max}(v, Q + 1) = p_v + P_{max}(v_n, Q + 1 - q_v)$ também seja máximo. Vale ressaltar que, caso $Q+1-q_v < 0$, teremos que $P_{max}(v_n, Q+1-q_v) = -\infty$, uma vez que é impossivel chegar a qualquer vértice consumindo um custo total menor que zero.

E assim então podemos, através da prova indutiva resolvida, podemos derivar um algoritmo genérico que corresponde a prova deste teorema. Segue abaixo então o algoritmo em **pseudocódigo** e em seguida o algoritmo em **Python**:

Implementação em Pseudocódigo - Algoritmo Genérico:

```
função caminhoDePremioMax(v, q)
    se q é zero
        se v é origem
            return caminho({v})
        caso contrario
            return "caminho impossivel"

    se q < 0
        return "caminho impossivel"

    Vn \(\Leftarrow\) conjunto de vizinhos de v
    caminho \(\Leftarrow\) maxPremio\{ caminhoDePremioMax( vn \(\in\) Vn, q-custo(v) )
    caminho.adicionaAoFinal(v)
    return caminho
```

Implementação em Python - Algoritmo Específico:

```python
def teo_3(pos, costs, rewards, energy):
  # Salvaguarda
  if pos < 0 or pos >= 64 or energy < 0:
    raise ValueError("Invalid position/energy!!")

  # Dict cujas chaves sao tuplas (posicao, energia) e cujos valores
  # sao tuplas contendo o premio maximo que o rei consegue coletar
  # comecando da posicao dada, com dada energia disponivel, e parando na
  # posicao 0 com 0 energia, e o caminho para tal
  memo = {}

  # Caso base -- q=0
  memo[(0, 0)] = (0, [0])
  for v in range(1, 64):
    memo[(v, 0)] = None

  # Hipotese indutiva e passo indutivo -- preencher a tabela
  # Para cada coluna de energia
  for q in range(1, energy+1):
    # Para cada vertice nessa coluna
    for v in range(64):
      # custo_vizinho e a coluna em que vamos olhar
      custo_vizinho = q - costs[v]
      vizinhos = find_neighbors(v)
      if custo_vizinho < 0:
        memo[(v, q)] = None
      else:
        # Filtra as celulas -- somente se nao for impossivel (not None)
        # e pega a tupla (premio, caminho) delas
```

```
        tuplas = [memo[(vizinho, custo_vizinho)] for vizinho in vizinhos if not memo[(vizinho, custo_vi
        if len(tuplas) > 0:
          # O melhor_vizinho e o que tem maior premio
          melhor_vizinho = max(tuplas, key=lambda x: x[0])
          # O novo_premio e o premio do melhor vizinho somado ao do vertice em questao
          novo_premio = melhor_vizinho[0] + rewards[v]
          # O novo_caminho e o caminho do melhor vizinho acrescido do vertice em questao
          novo_caminho = melhor_vizinho[1][:] + [v]
          memo[(v, q)] = (novo_premio, novo_caminho)
        else:
          memo[(v, q)] = None

  # Com a tabela em maos, vamos encontrar o caminho comecando em 0
  # que obtenha o maior premio possivel, utilizando qualquer quantidade
  # de energia menor que a fornecida
  maior = 0
  for q in range(energy, -1, -1):
    x = memo[(0, q)]
    if x != None and x[0] > maior:
      maior = x[0]
      inst = x + (q,)
  return inst


def find_neighbors(pos):
  x = pos/8
  y = pos%8

  naive = [(x-1, y-1), (x-1, y), (x-1, y+1),
           (x,   y-1),           (x,   y+1),
           (x+1, y-1), (x+1, y), (x+1, y+1)]
  filtered = [n for n in naive if n[0]>=0 and n[1]>=0 and n[0]<8 and n[1]<8]

  return map(lambda pos: pos[0]*8+pos[1], filtered)
```

Testes do Algoritmo: Os testes se encontram no arquivo Jupyter Notebook juntamente com o tempo de execução. Os testes foram realizados com instâncias pré-definidas que se encontram no arquivo *walk.in*. Os resultados encontram-se abaixo em uma tabela para melhor visualização. Segue também em anexo o arquivo walk.out.

Instância	Tempo	Prêmio Obtido	Energia Utilizada	Energia Disponível	Caminho Encontrado
1ª	3.70ms	16	8	8	0 1 0 1 0 1 0 1 0
2ª	3.52ms	16	8	8	0 1 0 1 0 1 0 1 0
3ª	3.97ms	16	8	8	0 1 0 1 0 1 0 1 0
4ª	14.22ms	284	22	22	0 9 18 27 36 44 52 60 52 60 52 60 52 60 52 60 52 44 36 27 18 9 0
5ª	9.28ms	57	18	18	0 8 16 25 16 25 16 25 16 25 16 25 16 25 16 25 16 8 0

2 Segunda Questão (Teorema 2)