Prova, Implementação e Análise de Problemas Selecionados II

Carlos Mattoso, Ian Albuquerque e Leonardo Kaplan

PUC-Rio

I. Introdução

Apresentou-se dois problemas com o objetivo de desenvolver códigos com diferentes estruturas de dados e algoritmos para solucioná-los, assim como analisar o desempenho das implementações destes algoritmos com respeito ao tempo de CPU. Neste trabalho procuramos expandir os conhecimentos empregados no primeiro, explorando a aplicação de tais técnicas para algoritmos em grafos.

É possível enunciar um teorema correspondente para cada um dos problemas descritos, o que permite descrevê-los, demonstrar seu entendimento em detalhe, explicar sua fundamentação e justificar sua corretude através de uma prova por indução matemática. Com base nesta, desenvolvemos um algoritmo e apresentamos dados a respeito de seu tempo de execução e dos resultados produzidos para um conjunto de instâncias.

As provas dos teoremas se baseiam principalmente no princípio de indução matemática de Peano, conforme descrito abaixo:

Se ϕ é um predicado unário tal que: $\phi(0)$ é verdade, e para todo número natural n, se ϕ(n) é verdadeiro, então ϕ(S(n)) também o é, então ϕ(n) é verdadeiro para todo número natural n.

Assim, para provar-se um teorema para diversos valores em um domínio discreto, basta provar-se os chamados Teoremas de Caso Base, em que prova-se o teorema para valores específicos e usualmente de simples entendimento do domínio, assim como os Teoremas do Passo Indutivo, em que prova-se que caso o teorema seja verdade para algum valor do domínio, o teorema também será verdade para outros valores do domínio diferentes do da hipótese.

Dessa forma, discriminamos para cada um dos três problemas seus teoremas correspondentes, cada um com seu enunciado, prova, algoritmo resultate, comentários e resultados.

Além disso, também empregaram-se as técnicas de reforço de hipótese e a aquela conhecida como preenchimento de tabelas ou programação dinâmica.

Dessa forma, segue-se o desenvolvimento de cada um dos problemas solicitados.

II. Problema 1

I. Enunciado

Teorema 1 : Sabe-se encontrar a árvore geradora máxima de um grafo.

II. Prova

A prova é feita por indução matemática utilizando o número de vértices como parâmetro de indução. O *teorema 1* pode ser enunciado:

Teorema vértice que liga a are1 (k): Sabe-se encontrar a árvore de peso máximo de $G = (V, E)$ que contém o vértice 1 e possui K vértices.

Teorema do Caso Base:

Teorema 1 (1):
Existe apenas um vértice v_1, portanto ele compõe unicamente a árvore. Àrvore geradora máxima: $V'_1 = v_1$, $E'_1 = \emptyset$. Peso da árvore geradora máxima: 0

Teorema do Passo Indutivo:

Teorema 1 (k) $\rightarrow$ ***Teorema 1 (k+1)***
Por hipótese indutiva conheço V'_k e E'_k.
Se considerar a componente conexa A formada por V'_k e E'_k e o subgrafo induzido B formado pelos vértices $\left(V - V'_k\right)$, necessariamente a aresta de maior peso entre A e B estará na árvore geradora máxima. As componentes resultantes possuirão

pelo menos $k+1$ vértices.

$E'_{(k+1)} = E'_k \cup argmax(\Gamma^+(V'_k))$
$V'_{(k+1)} = V'_k \cup$ vértice que liga a aresta que foi adicionada

III. Resultados

Entrada	Número de execuções	Tempo total	Tempo/execução
att48	12926	5.000000	0.000387
bays29	44611	5.000000	0.000112
berlin52	10895	5.000000	0.000459
bier127	1011	5.000000	0.004946
dantzig42	19633	5.000000	0.000255
eil51	12616	5.000000	0.000396
eil76	4169	5.000000	0.001199
lin105	1402	5.000000	0.003566
lin318	74	5.031250	0.067990
ulysses16	269782	5.000000	0.000019
ulysses22	117266	5.000000	0.000043

IV. Algoritmo Resultante

```
// Cada iteracao adiciona um vertice na arvore
  for(int i=0;i<num_vertices;i++)
  {
    // Encontrar a aresta de maior peso

    for(int vertex=0; vertex < num_vertices; vertex++)
    {
      if(is_vertice_in_tree[vertex])
      {
        // Percorre todo grafo
        for(int j=0;j < adjacency_list[vertex].size();j++)
        {
```

```
        if(edge_weight > max_weight &&
            !is_vertice_in_tree[vertex_checked])
        {
          max_weight = edge_weight;
          vertex_to_add = vertex_checked;
          origin_vertex = vertex;
        }
      }
    }
    else
    {
      continue;
    }
  }
  if(vertex_to_add == -1 || origin_vertex == -1)
  {
    break;
  }
  is_vertice_in_tree[vertex_to_add] = true;
  tree[vertex_to_add] = origin_vertex;
}
return tree;
```

As imagens exibindo o funcionamento do algoritmo para uma determinada instância encontram-se na sua respectiva pasta (*prim*), dentro da pasta base **src**.

III. Problema 1 - Desafio

I. Enunciado

Teorema 2 Sabe-se encontrar a floresta de peso mínimo de $G = (V, E)$ onde os com- ponentes conexos possuem pelo menos K vértices.

II. Prova

Teorema do Caso Base:

Teorema 1 (1):
Existe apenas um vértice v_1. Componentes conexos = cada vértice do grafo. Para cada componente conexa: $V' = \{v\}$ e $E' = \emptyset$ Peso da floresta geradora mínima: 0

Teorema do Passo Indutivo:

Teorema 1 (k) $\rightarrow$ ***Teorema 1 (k+1)***

Para cada componente conexa a formada por um subconjunto de V'_k e de E'_k, consideremos o subgrafo induzido B formado pelos vértices $\left(V - V'_k\right)$. Necessariamente a aresta de menor peso entre a e B estará na floresta geradora mínima. As componentes resultantes possuirão pelo menos $k+1$ Vértices.

$E'_{(k+1)} = E'_k \cup argmax(\Gamma^+(V'_k))$

$V'_{(k+1)} = V'_k \cup$ vértice que liga a aresta que foi adicionada

III. Resultados

Entrada	Número de execuções	Tempo total	Tempo/execução
att48	368306	5.000000	0.000014
bays29	953244	5.000000	0.000005
berlin52	318617	5.000000	0.000016
bier127	59383	5.000000	0.000084
dantzig42	463653	5.000000	0.000011
eil51	326307	5.000000	0.000015
eil76	158460	5.000000	0.000032
lin105	84935	5.000000	0.000059
lin318	9793	5.000000	0.000511
ulysses16	2563521	5.000000	0.000002
ulysses22	1526119	5.000000	0.000003
usa13509	2	5.671875	2.835938

IV. Algoritmo Resultante

```
pair<int,int>* forest = new pair<int,int>[num_vertices-1];
*num_edges_generated = 0;
int* conex_component = new int[num_vertices];
int number_of_conex_components = num_vertices;
bool has_merged = true;
```

```
// Caso Base (Componentes Conexos de Um Vertice)
for(int i=0;i < num_vertices;i++)
{
  conex_component[i]=i;
}

// Inducao

// Assumindo um grafo conexo, quero rodar o algoritmo ate sobrar 1
    componente conexa
while(has_merged)
{
  has_merged = false;
  for(int k=0;k < number_of_conex_components;k++)
  {
    // Encontrar a aresta de maior peso
    int min_weight = highest_weight_in_graph+1;
    int vertex_to_add = -1;
    int origin_vertex = -1;
    for(int vertex=0;vertex<num_vertices;vertex++)
    {
      if(conex_component[vertex]==k)
      {
        // Percorre todo o grafo
        for(int j=0;j < adjacency_list[vertex].size();j++)
        {
          int vertex_checked = adjacency_list[vertex][j].first;
          float edge_weight = adjacency_list[vertex][j].second;

          if(edge_weight<min_weight && conex_component[vertex_checked] !=
              conex_component[vertex])
          {
            min_weight = edge_weight;
            vertex_to_add = vertex_checked;
            origin_vertex = vertex;
          }
        }
      }
      else
      {
```

```
        continue;
      }
    }
    if(vertex_to_add != -1 && origin_vertex != -1)
    {
      has_merged = true;
      forest[*num_edges_generated] =
          make_pair(origin_vertex,vertex_to_add);
      (*num_edges_generated)++;
      int lowest_con_comp =
          conex_component[origin_vertex]<conex_component[vertex_to_add]?
              conex_component[origin_vertex]:conex_component[vertex_to_add];
      int highest_con_comp =
          conex_component[origin_vertex]<conex_component[vertex_to_add]?
              conex_component[vertex_to_add]:conex_component[origin_vertex];

      for(int vertex=0;vertex<num_vertices;vertex++)
      {
        if(conex_component[vertex]==highest_con_comp)
        {
          conex_component[vertex] = lowest_con_comp;
        }
      }
    }
  }
}
return forest;
```

As imagens exibindo o funcionamento do algoritmo para uma determinada instância encontram-se na sua respectiva pasta (*boruvka*), dentro da pasta base **src**.

IV. Problema 2

I. Enunciado

Seja um tabuleiro de xadrez n por m e um rei que está inicialmente na posição (1,1). Para cada posição do tabuleiro estão associados um prêmio p_ij e um consumo q_ij. Os prêmios e os consumos assumem somente valores positivos. O rei tem inicialmente Q unidades para consumir e pode passar quantas vezes quiser em cada posição do tabuleiro e a cada vez receber o prêmio e, naturalmente, consumir

os seus recursos. Ao final (do passeio) o rei tem que estar de volta na posição (1, 1).

Teorema 3$(\mathbf{i}, \mathbf{j}, \mathbf{q})$ Sabe-se determinar o prêmio máximo que o rei consegue coletar estando na posição (i,j) com q unidades restantes para consumir.

II. Prova

Para obtermos um reforço de hipótese, reescrevemos da seguinte forma o teorema:

Teorema 3$(\mathbf{x}, \mathbf{y}, \mathbf{q})$ Sabe-se determinar o caminho de prêmio máximo que o rei consegue coletar estando na posição (x,y) com q unidades restantes para consumir.

Teorema(s) do(s) Caso(s) Base(s):

Teorema 3 (0):
Se $(x, y) = (1, 1) \rightarrow P_{max}(x, y, q) = 0$ e $caminho = \emptyset$
Caso contrário $\rightarrow P_{max}(x, y, q) = -\infty$ e $caminho = \nexists$ pois não há como chegar em (1,1)

Teorema do Passo Indutivo:

Teorema 3 (x,y,k) $\forall k < q \rightarrow$ ***Teorema 3 (k+1)***

$V = Vizinhos = \{(i-1, j-1), (i, j-1), (i+1, j-1), (i-1, j), (i+1, j), (i-1, j+1), (i, j+1), (i, j+1), (i+1, j+1)\}$

$P_{max}(x, y, q) = max_{v \in V}\{Prmio(v) + P_{max}(v, q - custo(v))\}$

III. Resultados

Caso de teste	Número de execuções	Tempo total	Tempo/execução
1	65071	5.000310	0.000077
2	1	5.000310	5.000310
3	1	5.000310	5.000310
4	1	5.000310	5.000310
5	1	5.000310	5.000310

IV. Algoritmo Resultante

```
void walkOfAKing()
  for(int energy=0;energy <= starting_energy;energy++)
    for(int x=0;x<dim_x;x++)
      for(int y=0;y<dim_y;y++)
        if(energy==0)
          walkOfAKingBaseCase(x,y,energy);
        else
          walkOfAKingInductiveStep(x,y,energy);
```

Caso base:

```
void walkOfAKingBaseCase(int x,int y,int energy)
{
  if(x==0 && y==0)
  {
    max_prize[x][y][energy] = 0;
    path[x][y][energy] = END;
  }
  else
  {
    max_prize[x][y][energy] = MINUS_INFINITY;
    path[x][y][energy] = NONE;
  }
}
```

Passo Indutivo:

```
void walkOfAKingInductiveStep(int x,int y,int energy)
{
  int max_prize_found_yet = MINUS_INFINITY;
  int direction_of_max_prize = NONE;

  int cell_x;
  int cell_y;
  int cell_prize;
  int cell_cost;
```

```
  int cell_max_prize;
  for(int direction=UP_LEFT;direction<=BOT_RIGHT;direction++)
  {
    cell_x = x + dx[direction];
    cell_y = y + dy[direction];

    if(isOutOfBounds(cell_x,cell_y))
    {
      continue;
    }

    cell_prize = prize[cell_x][cell_y];
    cell_cost = cost[cell_x][cell_y];

    if(cell_cost > energy)
    {
      continue;
    }

    cell_max_prize =
        add(cell_prize,max_prize[cell_x][cell_y][energy-cell_cost]);

    if(cell_max_prize > max_prize_found_yet)
    {
      max_prize_found_yet = cell_max_prize;
      direction_of_max_prize = direction;
    }
  }

  max_prize[x][y][energy] = max_prize_found_yet;
  path[x][y][energy] = direction_of_max_prize;
}
```

Disponibilizamos em anexo uma implementação de um programa que exibe o percurso ideal do rei pelo tabuleiro, sendo atualizados o valor do prêmio obtido e a quantidade de energia após passar por cada posição. Além disso, na pasta do Walk of a King também apresentamos uma imagem com a ideia geral do funcionamento do algoritmo.

V. Conclusão

Todos os algoritmos resultantes das provas por indução conseguiram resolver os problemas apresentados. Contudo, para valores de seus respectivos parâmetros que produzam grandes resultados, todos os algoritmos acabaram pecando em termos de desempenho, principalmente em uso de memória.