Trabalho 2 – Estruturas Discretas
Lucas Hardman – 1113567
Stephanie Fay – 1121598

Obs: Utilizada versão 2.7 do Python

Exercício 1

(a)

Denomina-se A(k) a árvore obtida com certo valor de k. Denominamos V(k) e E(k) as listas de vértices e arestas, respectivamente, que compõem a árvore A(k).

Caso Base:

Para k = 1, temos apenas uma única árvore possivel, com 1 vértice, nenhuma aresta e peso total zero.

Passo Indutivo:

Pela hipótese indutiva temos que o teorema é válido para k vértices e vamos provar que também é válido para k + 1 vértices. Ou seja, conhecendo V(k) e E(k), desejamos determinar V(k+1) e E(k+1).

Considere um grafo B(k) formado pelos vértices V – V(k) e por todas as arestas formadas por vértices contidos em V – V(k). Conderando um conjunto C de arestas do tipo (a,b) onde 'a' pertence a A(k) e 'b' pertence a B(k), necessáriamente A(k+1) tem seu conjunto de vértices definido por V(k) U {b} e seu conjunto de arestas por E(k) U {(a,b)}.

'a' e 'b' são os vértices de aresta de maior peso entre as arestas de C, logo determinamos A(k+1) e com isso provamos o teorema.

(b)

Find(G, K)

Se K == 1

Return G #G contém 1 vértice

A <- Find(G, K-1)

B <- Grafo com os vertices de G não presentes em A e por todas as arestas entre estes vértices.

R <- Arestas do tipo (a,b) onde 'a' pertence a A e 'b' pertence a B.

Maior_peso <- elemento de R com maior peso

a,b <- vértices ligados pela aresta Maior_peso

Adiciona o vértice 'b' ao grafo A

Adiciona a aresta (a,b) ao grafo A

Return A

(c)

Consultar o arquivo exercicio1/teorema1.py

(d)

Instância	**K=1**	**K=2**	**K=3**	**K=10**	**K=25**	**K=40**
Att48	0.007338	0.133587	0.348467	4.093188	20.348667	36.255425
Bays29	0.007588	0.085334	0.212285	2.312006	8.427586	K > |V|
Bier127	0.008982	0.296900	0.85551	11.783915	77.515584	188.817996
Dantzig42	0.007460	0.120050	0.299805	3.562471	16.079755	27.447995
Eil51	0.007931	0.130993	0.353515	4.278844	22.432433	43.532956
Eil76	0.007815	0.186954	0.542641	6.763207	37.704000	86.321000
Lin105	0.008621	0.251181	0.712944	9.424548	56.425448	134.705023
Ulysses16	0.007054	0.054906	0.139679	1.064515	K > |V|	K > |V|

DESAFIO:

(e.a)

Caso Base:

Para k = 1, a florestas F(1) conterá todos os vértices de V, porém nenhuma aresta. Assim, haverá |V| componentes conexas e a soma dos pesos será mínima.

Passo Indutivo:

Pela hipótese indutiva temos que o teorema é válido para k vértices e vamos provar que também é válido para k + 1 vértices. Um componente conexo de F(k) possuirá pelo menos k vértices. Sendo assim, o único modo de garantir que este componente passe a conter pelo menos K + 1 vértices é adicionando um vértice a este componente.

Então, enquanto houverem componentes conexo em F(k+1) com números de vértices menores que k + 1 devemos escolher a aresta de menor peso do conjunto de arestas G ainda não utilizados em F(K+1) e passar para um componente conexo A de F(k+1), tal que apenas um vértice desta aresta pertença a A.

Desta maneira provamos o teorema, pois todo componente conexo contará com pelo menos k + 1 vértices e, assim, obteremos F(k+1).

(e.b)

```
Find(V, E, K)
    Se K == 1
        F <- floresta contendo todos os vértices e nenhuma aresta
        Return F
    F <- find (V, E, K-1)
    Enquanto houver componente conexa de F com número de vértices < K, faça
        C <- uma componente conexa qualquer de F
        E <- aresta mínima qualquer que não pertença a F com um vértice em F
        Adicionar E a F
    Return F
```

(e.c)

Consultar o arquivo exercicio1/teorema2.py

(e.d)

Instância	K=1	K=2	K=3	K=10	K=25	K=40
Att48	0.046065	117.829583	162.872937	338.602209	572.796106	571.550767
Bays29	0.033377	19.459214	28.347683	39.8448242	45.051225	K > |V|
Bier127	0.112639	5218.7669	8483.6828	18052.47688	28112.1060	28736.4180
Ulysses16	0.021745	3.160977	3.972414	7.972414	K > |V|	K > |V|

Exercício 2

(a)

Vamos considerar uma matriz de 64 vértices distintos, com v(1) = (1,1), v2 = (1,2) e assim sucessivamente, distribuída em uma tabela cujas linhas correspondem aos vértices e as colunas ao custo q disponivel para ser utilizado. A células da tabela serão preenchidas com o prêmio máximo Pmax(V(i,j), q) que se consegue a partir de um trajeto que inicie no vértice V(i,j) e que consuma q unidades.

Caso Base:

Para q = 1, preenchemos a primeira coluna da tabela. Neste caso, não existem unidades para consumir, logo não poderemos sair da origem (i,j). Ou seja, o prêmio máximo para ir até (i,j) será zero e para qualquer outro vértice será impossível.

Passo Indutivo:

Por indução forte, como hipótese indutiva temos que o teorema é válido para 0 <= q <= Q, e queremos provar que é válido também para Q + 1. 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 Pmax(v(n),Q+1-q(v)) seja máximo, garantindo assim que Pmax(v, Q+1) = p(v) + Pmax(v(n),Q+1-q(v)) também seja máximo. Vale ressaltar que caso Q + 1 - q(v) < 0, teremos que Pmax(v(n),Q+1-q(v)) é impossível.

(b)

MelhorCaminho(v, q)

Se q é 0, então

Se v é origem, então

Return caminho(v)

Senão,

Return “Caminho impossível”

Senão,

Return “Caminho impossível”

Vn <- conjunto de vizinhos de v

caminho <- maxPremio(MelhorCaminho(v(n) pertencente a V(n), q – custo(v)))

Adiciona caminho ao final de v

Return caminho

(c)

Consultar o arquivo exercicio2/teorema3.py

```
num = 0
time/exec = 6.517500 ms
num = 1
time/exec = 6.470000 ms
num = 2
time/exec = 6.627500 ms
num = 3
time/exec = 20.006667 ms
num = 4
time/exec = 16.310000 ms
16
(16, [0, 1, 0, 1, 0, 1, 0, 1, 0])
0
0 1 0 1 0 1 0 1 0
16
(16, [0, 1, 0, 1, 0, 1, 0, 1, 0])
0
0 1 0 1 0 1 0 1 0
16
(16, [0, 1, 0, 1, 0, 1, 0, 1, 0])
0
0 1 0 1 0 1 0 1 0
284
(284, [0, 9, 18, 27, 36, 44, 52, 60, 52, 60, 52, 60, 52, 60, 52, 60, 52, 44, 36, 27, 18, 9, 0])
0
0 9 18 27 36 44 52 60 52 60 52 60 52 60 52 60 52 44 36 27 18 9 0
57
(57, [0, 8, 16, 25, 16, 25, 16, 25, 16, 25, 16, 25, 16, 25, 16, 25, 16, 8, 0])
0
0 8 16 25 16 25 16 25 16 25 16 25 16 25 16 25 16 8 0
```

Instância	Tempo	Caminho
1ª	6.517500	0,1,0,1,0,1,0,1,0
2ª	6.470000	0,1,0,1,0,1,0,1,0
3ª	6.627500	0,1,0,1,0,1,0,1,0
4ª	20.006667	0,9,18,27,36,44,52,60,52,60,52,60,52,60,52,60,52,44,36,27,18,9,0
5ª	16.310000	0,8,16,25,16,25,16,25,16,25,16,25,16,25,16,25,16,8,0

Exercício 3

Levamos em consideração que o que foi pedido na questão foi para encontrar substrings. Ou seja, não precisa ser a palavra exata, ela pode ser diferente no inicio ou no fim (ou nos dois). Por exemplo, a palavra “cidade” dentro de “atrocidade”

(a)

```
aux = 0
while (args[x][0].find(args[x][1][y], aux) > 0):
        aux = args[x][0].find(args[x][1][y], aux)
        positions.append(aux)
        aux = aux + 1

print("Foram achadas " + str(len(positions)) + " correspondencias.")
```

(b)

```
args[x][0] = args[x][0].upper()
args[x][1][y] = args[x][1][y].upper()
aux = 0
while (args[x][0].find(args[x][1][y], aux) > 0):
        aux = args[x][0].find(args[x][1][y], aux)
        positions.append(aux)
        aux = aux + 1

print("Foram achadas " + str(len(positions)) + " correspondencias.")
```

(c)

```
temp = []
print("case sensitive:")
for i in range (0, len(args[x][1][y])):
        aux = list(args[x][1][y])
        aux.pop(i)
        temp.append("".join(aux))

for i in range (0, len(temp)):
```

```
		aux = 0
		positions = []
		while (args[x][0].find(temp[i], aux) > 0):
			aux = args[x][0].find(temp[i], aux)
			positions.append(aux)
			aux = aux + 1

		print("Foram achadas " + str(len(positions)) + " correspondencias para " + temp[i])

text = args[x][0].upper()
args[x][1][y] = args[x][1][y].upper()
temp = []
print("sem case sensitive:")
for i in range (0, len(args[x][1][y])):
		aux = list(args[x][1][y])
		aux.pop(i)
		temp.append("".join(aux))

for i in range (0, len(temp)):
		aux = 0
		positions = []
		while (text.find(temp[i], aux) > 0):
			aux = text.find(temp[i], aux)
			positions.append(aux)
			aux = aux + 1
```