Complexidade de Algoritmos

Mariana Kolberg

Projeto de Algoritmos
Algoritmos Gulosos

Aula 1

Algoritmos Gulosos

São úteis em problemas de otimização combinatória, onde a solução pode ser alcançada por uma seqüência de decisões.

Podemos aplicá-los, por exemplo, em:

- intercalação sucessiva ótima de listas;
- caminhos de custo mínimo em grafos orientados;
- árvore geradora (ou de espalhamento) mínima em grafos não orientados;
- escalonamento de tarefas
- balanceamento de carga entre processadores;
- etc.

Complexidade de Algoritmos

Intercalação sucessiva de listas.

- Consiste em intercalar **n listas**.
- Há várias maneiras possíveis de se realizar uma seqüência de intercalações.
- Cada seqüência está associada a um número de comparações.
- A escolha na ordem das intercalações afeta o desempenho do algoritmo.

Complexidade de Algoritmos

Considere duas listas L_1 e L_2 com comprimentos m_1 e m_2 , respectivamente.

O número de elementos da lista resultante da intercalação destas listas é igual a soma dos seus comprimentos, ou seja, $m_1 + m_2$

A quantidade de comparações necessárias para intercalar duas listas, no pior caso, é o comprimento da lista resultante menos 1, ou seja, $\mathbf{m_1 + m_2 - 1}$.

Sejam três listas L_1 , L_2 e L_3 com comprimentos 15, 10 e 5, respectivamente.

Qual é a seqüência ótima de intercalações ?

Complexidade de Algoritmos

1ª Maneira
$(L_1 + L_2) + L_3$

Intercalação $L_1 + L_2$

$|\ {}_1L_2\ | = 15+10 = 25$; comparações$({}_1L_2) = 24$

Intercalação ${}_1L_2 + L_3$

$|{}_{12}L_3| = 25+5 = 30$; comparações$({}_{12}L_3)=29$

Número total de comparações: 53

Complexidade de Algoritmos

2ª Maneira

$L_1 + (L_2 + L_3)$

Intercalação $L_2 + L_3$

$|\ {}_2L_3\ | = 10+5 = 15$; comparações$({}_2L_3) = 14$

Intercalação ${}_2L_3 + L_1$

$|{}_{23}L_1| = 15+15=30$; comparações$({}_{23}L_1)=29$

Número total de comparações: 43

Complexidade de Algoritmos

3ª Maneira
$(L_1 + L_3) + L_2$

Intercalação $L_1 + L_3$

$|\ {}_1L_3\ | = 15+5 = 20$; comparações$({}_1L_3) = 19$

Intercalação ${}_1L_3 + L_2$

$|{}_{13}L_2| = 20+10$; comparações$({}_{12}L_3)=29$

Número total de comparações: 48

Podemos tirar alguma conclusão ?

A ordem em que as intercalações são realizadas afeta substancialmente o número de comparações, ou seja, afeta o desempenho do algoritmo.

Complexidade de Algoritmos

Considere **3** listas $\mathbf{L_i}$ com comprimentos $\mathbf{m_i}$, para i = 1, 2, 3. Qual é influência da seqüência de intercalações deste conjunto no número de comparações realizadas?

Assuma que ${}_jm_k = m_j + m_k$. Vamos considerar duas situações

a) $(L_1 + L_2) + L_3$,
número total de comparações é igual a

$$t' = {}_1m_2 - 1 + ({}_1m_2 + m_3 - 1) = 2 . (m_1 + m_2) + m_3 - 2.$$

b) $(L_2 + L_3) + L_1$
o número total de comparações é igual a

$$t'' = {}_2m_3 - 1 + (m_1 + {}_2m_3 - 1) = 2 . (m_2 + m_3) + m_1 - 2$$

Complexidade de Algoritmos

Podemos afirmar que se $_1m_2 \leq {}_2m_3$, i. e., $(m_1 + m_2) \leq (m_2 + m_3)$, teremos t' $\leq$ t"?

(sendo t' =2 . (m_1 + m_2) + m_3 -2 e t" = 2 . (m_2+ m_3) + m_1-2)

Complexidade de Algoritmos

Podemos afirmar que se $_1m_2 \leq {}_2m_3$, i. e., ($m_1 + m_2$) $\leq$ (m_2+ m_3), teremos t' $\leq$ t"**?**
(t' começa com L_1 e L_2 e t" com L_2 e L_3)

$$(m_1 + m_2 \leq m_2 + m_3) = (m_1 + \textcircled{m_1} + m_2 \leq m_2 + m_3 + \textcircled{m_1}) =$$

$$(m_1 + m_1 + m_2 + \textcircled{m_2} \leq m_2 + \textcircled{m_2} + m_3 + m_1) =$$

$$(m_1 + m_1 + m_2 + m_2 + \textcircled{m_3} \leq m_2 + m_2 + \textcircled{m_3} + m_3 + m_1) =$$

$$(m_1 + m_1 + m_2 + m_2 + m_3 \textcircled{-2} \leq m_2 + m_2 + m_3 + m_3 + m_1 \textcircled{-2})$$

$$2(m_1 + m_2) + m_3 - 2 \leq 2(m_2 + m_3) + m_1 - 2$$

$$t' \leq t''$$

Complexidade de Algoritmos

Considere a intercalação de n listas L_i

Instante 1. A intercalação de L_1 com L_2 necessita de $_1\mathbf{m_2}\mathbf{-1}$ comparações e produz $_1L_2$ com comprimento $_1m_2$, onde $_1m_2 := (m_1 + m_2)$

Instante i. A intercalação de $_1L_i$ com L_{i+1} usa $_1m_{i+1}$-1comparações e produz $_1L_{i+1}$ com comprimento $_1m_{i+1}$, onde $_1m_{i+1} := (_1m_i + m_{i+1})$

Instante n-1. A intercalação de $_1L_{n-1}$ com L_n, usa $_1m_n$-1 comparações e produz $_1L_n$ com comprimento $_1m_n = (_1m_{n-1} + m_n)$

Qual é o número total de comparações realizadas ?

Complexidade de Algoritmos

Considere a intercalação de n listas L_i

Instante 1. A intercalação de L_1 com L_2 necessita de $\mathbf{{}_1m_2-1}$ comparações e produz ${}_1L_2$ com comprimento ${}_1m_2$, onde ${}_1m_2 := (m_1 + m_2)$

Instante i. A intercalação de ${}_1L_i$ com L_{i+1} usa ${}_1m_{i+1}$-1comparações e produz ${}_1L_{i+1}$ com comprimento ${}_1m_{i+1}$, onde ${}_1m_{i+1} := ({}_1m_i + m_{i+1})$

Instante n-1. A intercalação de ${}_1L_{n-1}$ com L_n, usa ${}_1m_n$-1 comparações e produz ${}_1L_n$ com comprimento ${}_1m_n = ({}_1m_{n-1} + m_n)$

Qual é o número total de comparações realizadas ?

O número total de comparações é dado por ${}_1m_2 + \ldots + {}_1m_{i+1} + \ldots + {}_1m_n - (n-1)$, ou seja,

$$(n-1).m_1 + (n-1)\, m_2 + (n-2)m_3 + \ldots + (n-i)m_{i+1} + \ldots + m_n-(n-1)$$

o primeiro par de listas intercalado dá a maior contribuição para este cálculo

Complexidade de Algoritmos

Considere 5 listas, com comprimentos 10, 20, 30, 40 e 50, a intercalar:

10 **20** 30 40 50

30 **30** 40 50

60 **40** **50**

60 **90**

150

Quantas comparações foram realizadas ?

Complexidade de Algoritmos

Considere 5 listas, com comprimentos 10, 20, 30, 40 e 50, a intercalar:

10 **20** 30 40 50

30 **30** 40 50

60 **40** **50**

60 **90**

150

Quantas comparações foram realizadas ?
326

Quantas comparações seriam realizadas se pegássemos
Sempre as duas maiores listas ?

Complexidade de Algoritmos

Considere 5 listas, com comprimentos 10, 20, 30, 40 e 50, a intercalar:

10 **20** 30 40 50

30 **30** 40 50

60 **40** **50**

60 **90**

150

Quantas comparações foram realizadas ?

326

Quantas comparações seriam realizadas se pegássemos
Sempre as duas maiores listas ?

496

Complexidade de Algoritmos

Algoritmo : Intercalação ótima de listas

Função Interc_Suc_Lst(L : D) → R

1. ncp ← 0 ;
2. repita
3. escolhe as duas menores listas L' e L" em L ;
4. L ← L – { L' , L" } ;
5. L* ← Interc(L' , L") ;
6. L ← L ∪ {L* } ;
7. ncp ← ncp + tam(L') + tam(L") – 1 ;
8. até-que |L| = 1 ;
9. L* ← a única lista em L ;
10. retorne-saída (L* , ncp) ;
11. fim-Função

Complexidade de Algoritmos

Idéias básicas

- Um algoritmo guloso seleciona, a cada passo, o melhor elemento pertencente a entrada.
- Verifica se ele é viável - vindo a fazer parte da solução ou não.
- Após uma seqüência de decisões, a solução do problema é alcançada.

Na seqüência de decisões, nenhum elemento é examinado mais de uma vez: ou ele fará parte da saída, ou será descartado.

Substrutura ótima: quando o problema contém em seu interior soluções ótimas para os subproblemas. A solução ótima local garante a solução ótima global.

No caso da intercalação, verifica sempre o par de listas de menor custo!

Complexidade de Algoritmos

A estratégia gulosa possui a seguinte estrutura geral

Inicialização ; Iteração ; Finalização.

- A Inicialização prepara a entrada (muitas vezes a entrada é classificada) e inicializa a saída.

- A Iteração
  - seleciona um elemento conforme uma função "gulosa",
  - marca-o para não considerá-lo novamente no futuro,
  - atualiza a entrada,
  - examina o elemento selecionado quanto sua viabilidade e decide a sua participação ou não na solução.

- A finalização recupera a saída.

Complexidade de Algoritmos

Algoritmo : Algoritmo Guloso
Função Alg_Gul(d:D)→R {Algoritmo Guloso (abstrato)}
{*Entrada-saída*: saída r:R é resposta ótima para entrada d:D}
Incz_G;Iter_G;Fnl_G {estrutura geral}
onde {componentes}
Incz_G: {inicialização}
inicializa parte da entrada a examinar e resposta parcial
Iter_G: itera corpo Crp_G {iteração}
seleciona elemento da parte da entrada a examinar;
remove elemento da parte da entrada a examinar;
inclui elemento na resposta parcial se viável
Fnl_G: {finalização}
recupera a saída a partir da resposta parcial

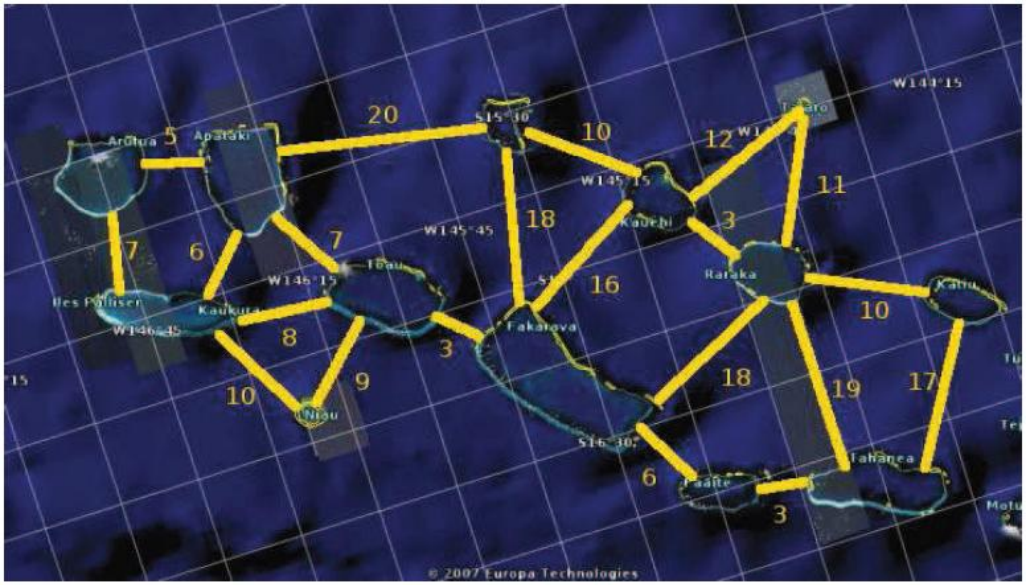
Complexidade de Algoritmos

Intercalação ótima de listas x Algoritmo Guloso

Interc_Suc_Lst	Alg_Gul	
sorte D := multiconjunto de listas	sorte D	{entrada}
1. ncp ← 0;	Incz_G	{inicialização}
2. repita	Iter_G	{iteração}
3. (L', L") ← 2 menores em L;	seleciona da parte da entrada;	
4. L ← L − {L', L"};		
5. L* ← Interc(L', L");	remove da parte da entrada;	
6. L ← L ∪ {L*};		
7. ncp ← ncp + tam(L') + tam(L") - 1;	inclui na resposta parcial se viável;	
8. até-que |L| = 1;		{fim da iteração}
9. L* ← a única lista em L;	Fnl_G	{finalização}

Complexidade de Algoritmos

- Árvore espalhada mínima
  - Uma árvore geradora (ou de espalhamento) de um grafo G é um subgrafo acíclico que contém todos os vértices do grafo.
  - Motivação: pontes para polinésia francesa



- Aplicações:
  - Redes elétricas
  - Sistemas de estradas
  - Pipelines
  - Caixeiro viajante

Complexidade de Algoritmos

A árvore geradora (ou de espalhamento) mínima

- É um problema de otimização:
  - O número de árvores espalhadas pode ser exponencial
  - Como achar a árvore de menor custo?

Usando as idéias básicas do método guloso

1. Inicialização: as arestas são ordenadas em ordem crescente de seus custos; e cada vértice é considerado uma árvore distinta. Isto gera uma floresta de árvores.
2. Iteração: a cada passo, seleciona-se uma aresta de custo mínimo (ainda não examinada). Se a inclusão desta aresta na configuração de árvores corrente criar um ciclo, ela é descartada; caso contrário, ela é incluída nesta configuração de tal forma que ligue duas árvores da floresta.
3. Finalização: retorna a árvore

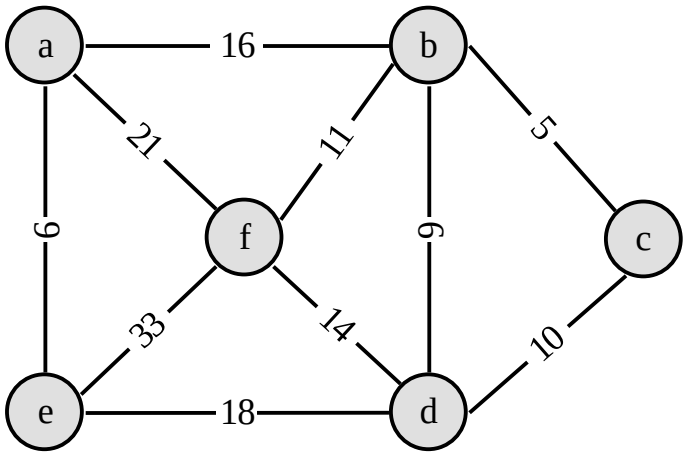
Qual é este algoritmo?

Complexidade de Algoritmos

Considere um grafo não orientado G = < V, E > com

- V = { a, b, c, d, e, f };

- E={(a, b),(a, e), (a, f), (b, c), (b, d), (b, f), (c, d), (d, e), (d, f), (e, f)};

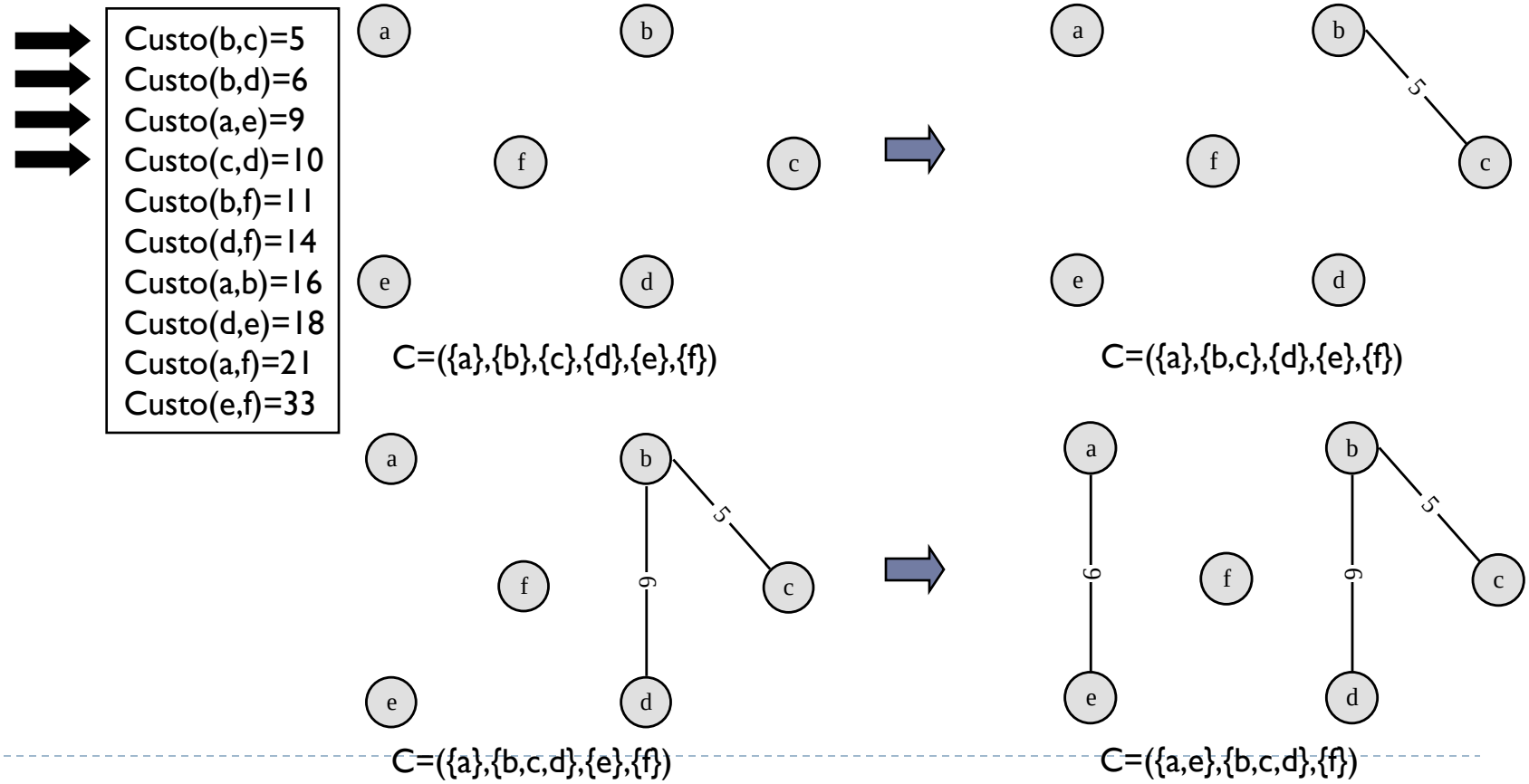


Custo	a	b	c	d	e	f
a	0	16	0	0	9	21
b	16	0	5	6	0	11
c	0	5	0	10	0	0
d	0	6	10	0	18	14
e	9	0	0	18	0	33
f	21	11	0	14	33	0

Complexidade de Algoritmos

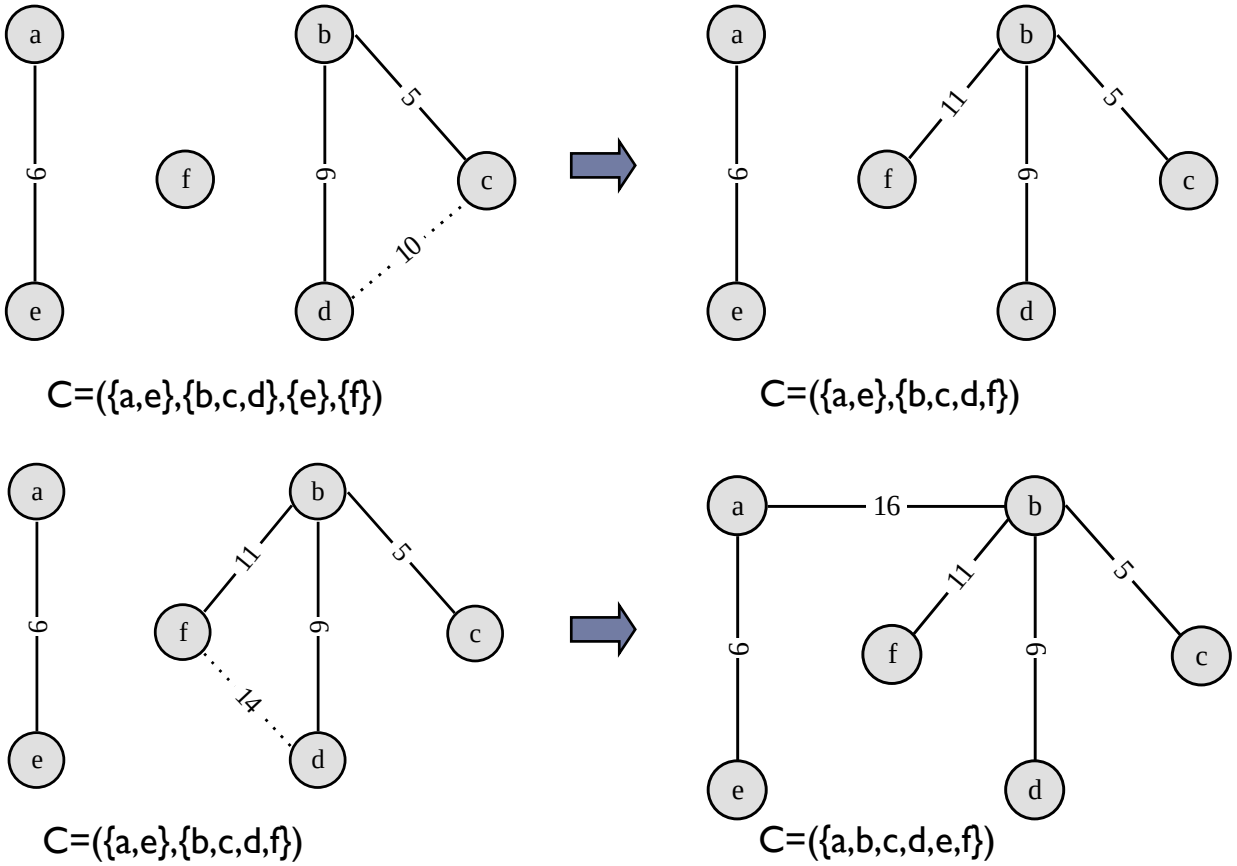
Projeto e Análise de Algoritmos

A construção da árvore espalhada mínima para G é como segue

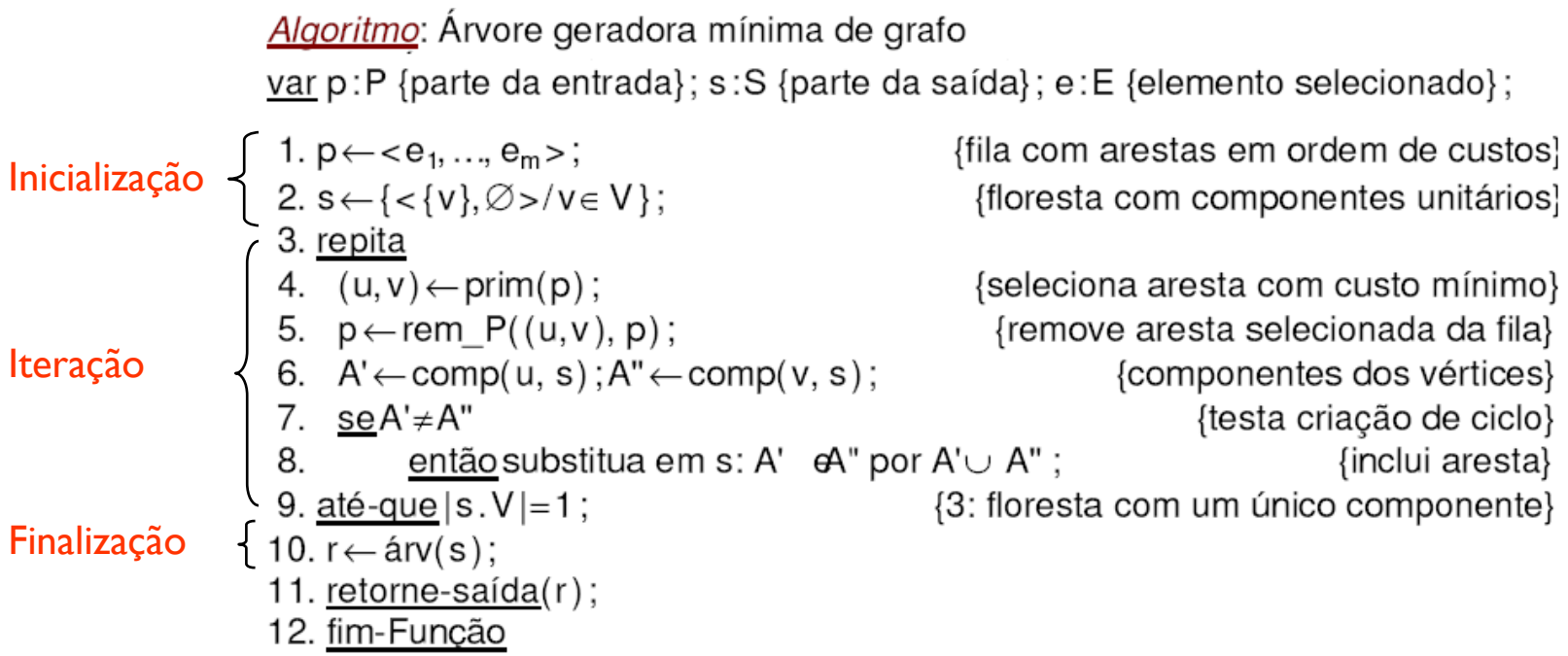


Complexidade de Algoritmos

Custo(b,c)=5
Custo(b,d)=6
Custo(a,e)=9
Custo(c,d)=10
Custo(b,f)=11
Custo(d,f)=14
Custo(a,b)=16
Custo(d,e)=18
Custo(a,f)=21
Custo(e,f)=33

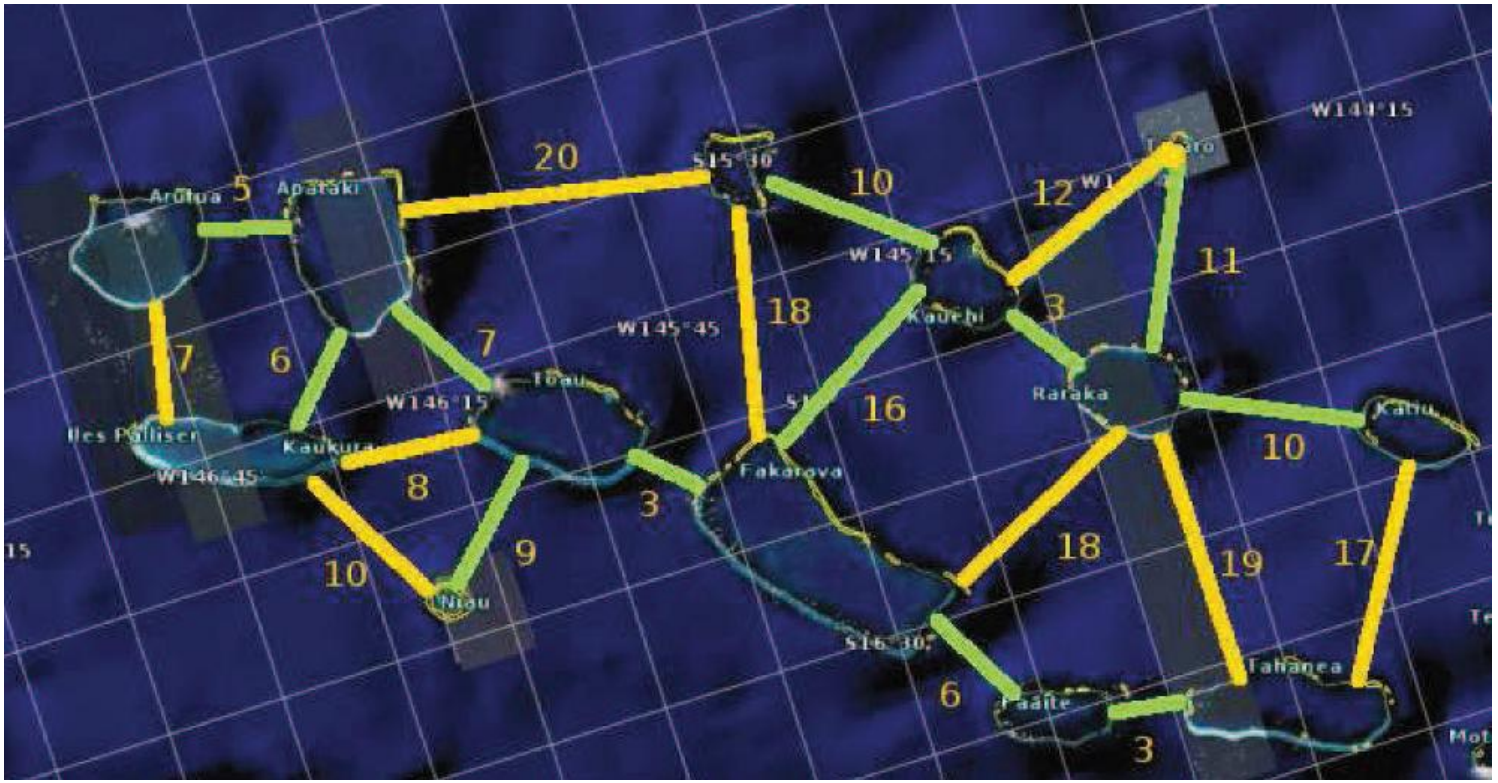


Complexidade de Algoritmos



Complexidade de algoritmos

- Resultado da aplicação do algoritmo para o problema das pontes na polinésia francesa.



Complexidade de Algoritmos

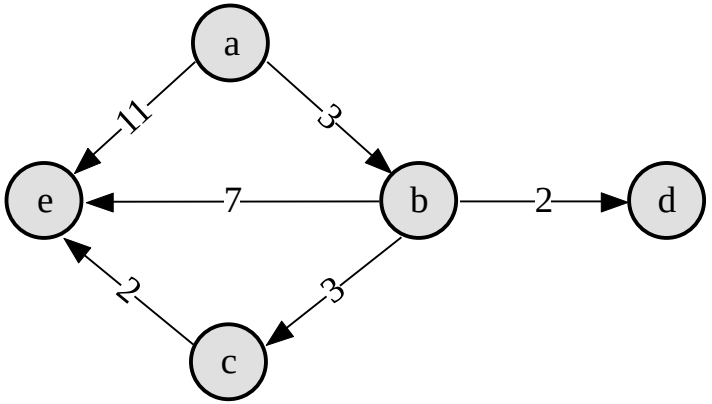
Caminhos de custo mínimo em grafo orientado

Este problema consiste em determinar um caminho de custo mínimo a partir de um vértice fonte a cada vértice do grafo.

Considere um grafo orientado G = < V, E > com 5 vértices: V = {a, b, c, d, e} e

6 arestas com a seguinte matriz de custos:

Custo	a	b	c	d	e
a	0	3	∞	∞	11
b	∞	0	3	2	7
c	∞	∞	0	∞	2
d	∞	∞	∞	0	∞
e	∞	∞	∞	∞	0



Complexidade de Algoritmos

- Substrutura ótima do caminho mais curto:
  - Considere o caminho mais curto de v1v2…vn
  - Tem substrutura ótima pois um subcaminho vi…vj de v1v2..vn também é o mais curto
    - Senão seria possível obter um caminho ainda mais curto!
- Substrutura ótima do caminho mais longo?

Complexidade de Algoritmos

O algoritmo constrói incrementalmente um conjunto I de vértices intermediários e o usa para calcular os caminhos de custo mínimo a partir do vértice fonte.

A cada passo, seleciona-se um novo vértice intermediário cuja distância em relação ao vértice fonte seja mínima (estratégia gulosa).

Este vértice intermediário é utilizado para atualizar o custo associado aos demais vértices em relação ao vértice fonte.

Quando **| I | = n**, temos todos os caminhos com custo mínimo a partir da fonte.

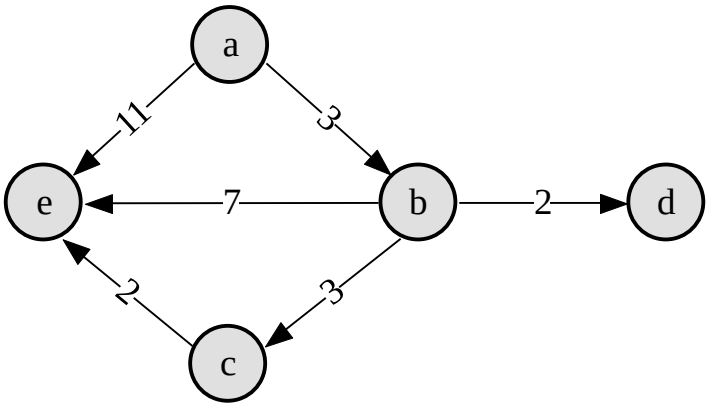
Importante: cada vértice é considerado apenas uma vez.

Complexidade de Algoritmos

Considere que o vértice fonte é o vértice **a**

Inicialmente, o conjunto I de vértices intermediários é vazio e o vetor de distâncias é inicializado com a primeira linha da matriz de custos.

A cada passo, seleciona-se um vértice v que tenha a menor distância em relação a fonte. Em seguida, atualiza-se a distância deste vértice em relação aos demais.



b	c	d	e
3	∞	∞	11

passo	I	vértice	b	c	d	e
1	∅	1: b	3	6	5	10
2	{b}	2: d	3	6	5	10
3	{b, d}	3: c	3	6	5	8
4	{b, c, d}	4: e	3	6	5	8

Complexidade de Algoritmos

Algoritmo: Custo mínimo de caminhos a partir de fonte em grafo orientado

Inicialização

0. $V_0 \leftarrow V - \{v_0\}$; {vértices não fonte: $v_1, \ldots, v_n$}
1. $p \leftarrow V_0$; {inicializa porção da entrada com $\{v_1, \ldots, v_n\}$}
2. para i de 1 até n faça dist[i] $\leftarrow$ custo[0, i]; {inicializa resposta parcial}

Iteração

3. para i de 1 até n faça {itera: 10}
4. e $\leftarrow$ vértice $v_j \in$ p com dist[j] mínimo; {seleciona novo vértice}
5. $p \leftarrow p - \{e\}$; {remove vértice selecionado}
6. para cada $v_i \in V_0$ faça {vértices $v_1, \ldots, v_n$: 9}
7. c $\leftarrow$ dist[j] + custo[v_j, v_i]; {distância usando v_j}
8. se c $\leq$ dist[i] então dist[i] $\leftarrow$ c ; {atualiza distância}
9. fim-para {6: cada $v_i \in V_0$}
10. fim-para {3: repita}

Finalização

11. retorne-saída(dist); {dá como saída resposta pronta}
12. fim-Função {fim do algoritmo Dist_fnt: distância a partir da fonte}

Cálculo da complexidade de algoritmos gulosos

Complexidade de Algoritmos

Complexidade de intercalação ótima de listas

O algoritmo efetua operações sobre o multiconjunto L, portanto a complexidade do algoritmo pode variar conforme a estrutura de dados escolhida para representar L.

O multiconjunto L é uma lista encadeada, em que cada elemento guarda tanto uma lista quanto seu tamanho. Assumiremos que

- para determinar as duas menores listas, a L é percorrida, com complexidade O(n);
- para remoção a complexidade é O(n);
- para a inclusão a complexidade é constante.

Complexidade de Algoritmos

Complexidade: Intercalação Ótima de Listas

Algoritmo : Intercalação ótima de listas

Função Interc_Suc_Lst(L : D) → R

Inicialização

1. ncp ← 0 ;

Iteração

2. repita
3. escolhe as duas menores listas L' e L" em L ;
4. L ← L – { L' , L" } ;
5. L* ← Interc(L' , L") ;
6. L ← L ∪ {L* } ;
7. ncp ← ncp + tam(L') + tam(L") – 1 ;
8. até-que |L| = 1 ;

Finalização

9. L* ← a única lista em L ;
10. retorne-saída (L* , ncp) ;
11. fim-Função

Complexidade de Algoritmos

O Algoritmo

recebe como entrada n listas (com seus comprimentos)

fornece como saída o par (intercalação das n listas em L, número mínimo de comparações).

A operação fundamental é a comparação entre elementos das listas e o tamanho da entrada é o par formado pelo tamanho n do multiconjunto L e pelo comprimento m da maior lista em L.

Complexidade de Algoritmos

Encontre a complexidade pessimista do algoritmo abaixo

Algoritmo : Intercalação ótima de listas

Função Interc_Suc_Lst(L : D) → R

Inicialização

1. ncp ← 0 ;

Iteração

2. repita
3. escolhe as duas menores listas L' e L" em L ;
4. L ← L – { L' , L" } ;
5. L* ← Interc(L' , L") ;
6. L ← L ∪ {L* } ;
7. ncp ← ncp + tam(L') + tam(L") – 1 ;
8. até-que |L| = 1 ;

Finalização

9. L* ← a única lista em L ;
10. retorne-saída (L* , ncp) ;
11. fim-Função

Complexidade de Algoritmos

O desempenho do algoritmo tem contribuições dadas por suas componentes:

Inicialização, Iteração e Finalização.

```
Algoritmo : Intercalação ótima de listas
Função Interc_Suc_Lst( L : D ) → R
Inicialização {
1. ncp ← 0 ;
2. repita
3.      escolhe as duas menores listas L' e L" em L ;
4.      L ← L – { L' , L" } ;
5.      L* ← Interc(L' , L") ;
6.      L ← L ∪ {L* } ;
7.      ncp ← ncp + tam(L' ) + tam(L") – 1 ;
8. até-que |L| = 1 ;
Finalização {
9. L* ← a única lista em L ;
10. retorne-saída ( L* , ncp ) ;
11. fim-Função
```

Como a inicialização e a finalização não envolvem comparações.

Logo,

desemp[Inicialização] = 0

desemp[Finalizacão] = 0.

Complexidade de Algoritmos

```
Algoritmo : Intercalação ótima de listas
Função Interc_Suc_Lst( L : D ) → R
1. ncp ← 0 ;
2. repita
3.       escolhe as duas menores listas L' e L" em L ;
4.       L ← L – { L' , L" } ;
5.       L* ← Interc(L' , L") ;
6.       L ← L ∪ {L* } ;
7.       ncp ← ncp + tam(L' ) + tam(L") – 1 ;
8. até-que |L| = 1 ;
9. L* ← a única lista em L ;
10. retorne-saída ( L* , ncp ) ;
11. fim-Função
```

Iteração

A cada iteração

- As duas menores listas em L são substituídas por sua intercalação
- O tamanho de L diminui de 1.

A iteração é executada para | L | variando de n a 1

i	0	1	…	k	…	n-1
$\|L_i\|$	n	n-1	…	n-k	…	1

Complexidade de Algoritmos

No ínicio da i-ésima iteração, $|L_i| = n - i + 1$. Portanto temos

comando		cota superior
3.	$(L', L'') \leftarrow$ 2 menores em L;	$n-i+1$
4.	$L \leftarrow L - \{L', L''\}$;	$n-i+1$
5.	$L^* \leftarrow$ Interc(L', L'');	compr(L') + compr(L'') - 1
6.	$L \leftarrow L \cup \{L^*\}$;	0
7.	ncp $\leftarrow$ ncp + compr(L') + compr(L'') - 1;	0

Sendo m'_i e m''_i os tamanhos das duas listas selecionadas na i-ésima iteração, temos

$$\text{desemp[Corpo-Iteração]} = n - i + 1 + n - i + 1 + m'_i + m''_i - 1$$

$$\text{desemp[Corpo-Iteração]} = 2(n - i) + m'_i + m''_i + 1$$

Complexidade de Algoritmos

Projeto e Análise de Algoritmos

O desempenho da iteração é

$$\text{desemp[Iteração]} = \sum_{i=1}^{n-1}(2(n-i) + m'_i + m''_i + 1)$$

$$= \sum_{i=1}^{n-1} 2(n-i) + \sum_{i=1}^{n-1}(m'_i + m''_i) + \sum_{i=1}^{n-1} 1$$

$$= 2\sum_{i=1}^{n-1}(i) + \sum_{i=1}^{n-1}(m'_i + m''_i) + (n-1)$$

$$= \frac{2(n-1).n}{2} + (n-1) + \sum_{i=1}^{n-1}(m'_i + m''_i)$$

Complexidade de Algoritmos

Tomando m como o máximo dos comprimentos das n listas em L, temos

$m'_i + m''_i \leq n.m$ Pq ?

$$= \sum_{i=1}^{n-1}(m'_i + m''_i) \leq \sum_{i=1}^{n-1}(nm) = (n-1).n.m$$

$$\text{desemp[Iteração]} \leq (n-1).n + (n-1) + (n-1).n.m$$

$$\text{desemp[Iteração]} \leq n^2 - n + n - 1 + n^2 m - n.m$$

$$\text{desemp[Iteração]} \leq n^2 - 1 + n^2 m - n.m$$

$$\text{desemp[Iteração]} \leq n^2 + n^2 m = n^2(m+1)$$

Complexidade de Algoritmos

O desempenho de Interc_Suc_Lst é

$$\text{desemp}[\text{Algoritmo}] \leq 0 + n^2(m+1) + 0$$

$$c_p^{\leq}[\text{Algoritmo}] = O(n^2(m+1)) = O(n^2.m)$$

Complexidade de Algoritmos

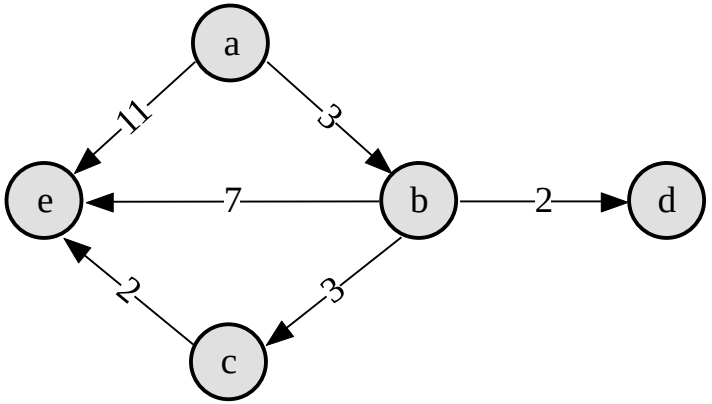
Caminhos de custo mínimo em grafo orientado

Este problema consiste em determinar um caminho de custo mínimo a partir de um vértice fonte a cada vértice do grafo.

Considere um grafo orientado G = < V, E > com 5 vértices: V = {a, b, c, d, e} e

6 arestas com a seguinte matriz de custos:

Custo	a	b	c	d	e
a	0	3	∞	∞	11
b	∞	0	3	2	7
c	∞	∞	0	∞	2
d	∞	∞	∞	0	∞
e	∞	∞	∞	∞	0



Complexidade de Algoritmos

Algoritmo: Custo mínimo de caminhos a partir de fonte em grafo orientado

Inicialização

0. $V_0 \leftarrow V - \{v_0\}$; {vértices não fonte: $v_1, \ldots, v_n$}
1. $p \leftarrow V_0$; {inicializa porção da entrada com $\{v_1, \ldots, v_n\}$}
2. para i de 1 até n faça dist[i] $\leftarrow$ custo $[v_0, v_i]$ {inicializa resposta parcial}

Iteração

3. para i de 1 até n faça {itera: 10}
4. e $\leftarrow$ vértice $v_j \in$ p com dist[j] mínimo; {seleciona novo vértice}
5. $p \leftarrow p - \{e\}$; {remove vértice selecionado}
6. para cada $v_i \in V_0$ faça {vértices $v_1, \ldots, v_n$: 9}
7. $c \leftarrow$ dist[j] + custo[v_j, v_i]; {distância usando v_j}
8. se $c \leq$ dist[i] então dist[i] $\leftarrow c$; {atualiza distância}
9. fim-para {6: cada $v_i \in V_0$}
10. fim-para {3: repita}

Finalização

11. retorne-saída(dist); {dá como saída resposta pronta}
12. fim-Função {fim do algoritmo Dist_fnt: distância a partir da fonte}

Complexidade de Algoritmos

O algoritmo

recebe como **entrada**

Um grafo orientado valorado G com **fonte v_0** e uma **matriz de custos**

fornece como **saída**

Um vetor dist (com os custos dos melhores caminhos a partir de v_0).

Consideremos um grafo orientado G com conjunto **$V = \{ v_0, v_1, \ldots, v_n \}$** de vértices.

As operações fundamentais do algoritmo são as manipulações com conjuntos (de vértices) e matrizes; e para o tamanho da entrada o número **n** de vértices não fonte.

Complexidade de Algoritmos

Encontre a copmplexidade pessimista do algoritmo abaixo, que tem contribuições dadas por suas componentes: **inicialização, iteração e finalização**.

Inicialização

0. $V_0 \leftarrow V - \{v_0\}$; {vértices não fonte: $v_1, \ldots, v_n$}
1. $p \leftarrow V_0$; {inicializa porção da entrada com $\{v_1, \ldots, v_n\}$}
2. para i de 1 até n faça dist[i] $\leftarrow$ custo[0,i]; {inicializa resposta parcial}

Iteração

3. para i de 1 até n faça {itera: 10}
4. e $\leftarrow$ vértice $v_j \in$ p com dist[j] mínimo; {seleciona novo vértice}
5. $p \leftarrow p - \{e\}$; {remove vértice selecionado}
6. para cada $v_i \in V_0$ faça {vértices $v_1, \ldots, v_n$: 9}
7. $c \leftarrow$ dist[j] + custo[v_j, v_i]; {distância usando v_j}
8. se $c \leq$ dist[i] então dist[i] $\leftarrow c$; {atualiza distância}
9. fim-para {6: cada $v_i \in V_0$}
10. fim-para {3: repita}

Finalização

11. retorne-saída(dist); {dá como saída resposta pronta}
12. fim-Função {fim do algoritmo Dist_fnt: distância a partir da fonte}