Problema 1

Exercício (CLRS 22-4.) Seja $G = (V, E)$ um grafo orientado onde cada vértice $u \in V$ tem um rótulo associado $L(u) \in \mathbb{Z}_+$. Para cada $u \in V$ seja $R(u)$ o conjunto dos vértices atingíveis por u. Seja

$$\min_L(u) = \min\{L(v) : v \in R(u)\}$$

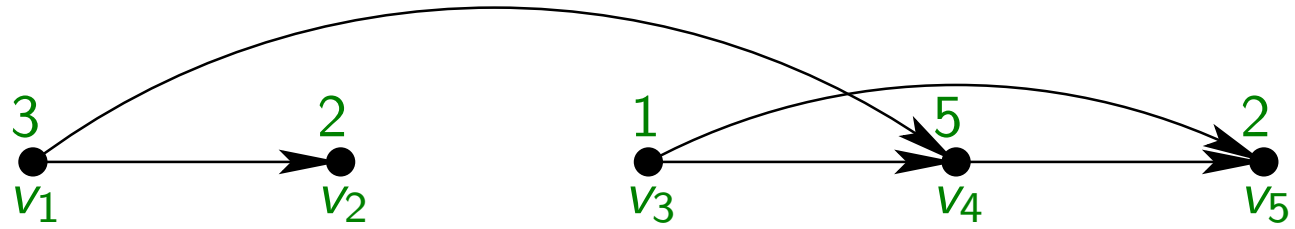
i.e., $\min_L(u)$ é o menor rótulo de um vértice atingível por u.

Descreva um algoritmo de complexidade $O(V + E)$ que dados G e L, calcula $\min_L(u)$ para todo $u \in V$.

Sugestão: como você resolveria o problema se G fosse **acíclico**?

Solução para G acíclico

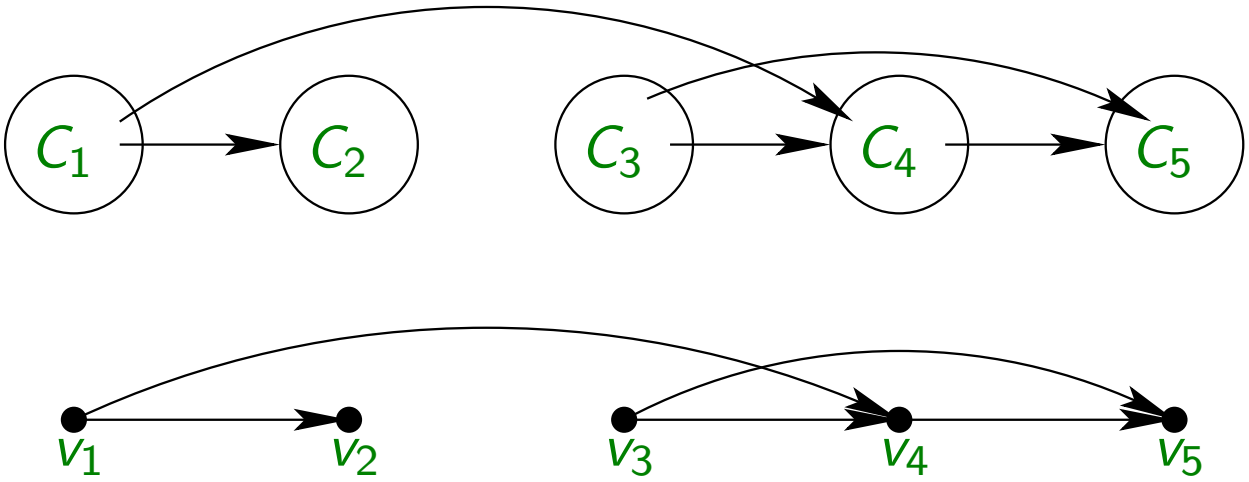
Suponha primeiro que $G = (V, E)$ é **acíclico**. Neste caso podemos resolver o problema usando Ordenação Topológica.



$\min_L(v_1) = 2$, $\min_L(v_2) = 2$, $\min_L(v_3) = 1$, $\min_L(v_4) = 2$ e $\min_L(v_5) = 2$.

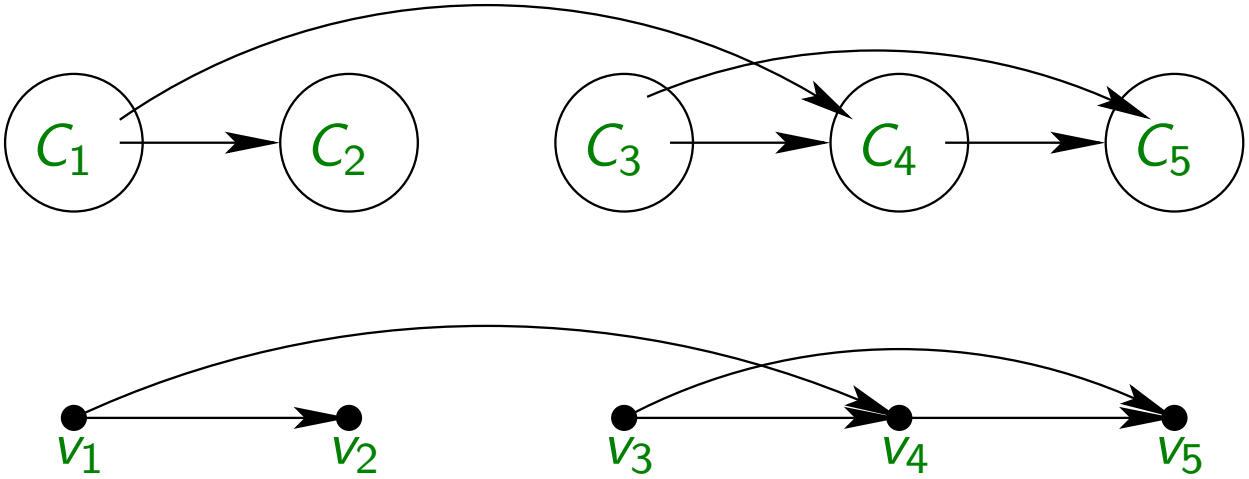
para cada $u \in V$ **faça** $\mathrm{MIN}_L[u] \leftarrow L[u]$
seja $v_1, v_2, \ldots, v_n$ uma ordenação topológica de G
para $i \leftarrow n$ **decrescendo até** 1 **faça**
  para cada $v \in \mathrm{Adj}[u]$ **faça**
    se $L(v) < \mathrm{MIN}_L[u]$ **então** $\mathrm{MIN}_L[u] \leftarrow L(v)$

Solução



- Vimos em aula um algoritmo que determina as componentes fortes (ou fortemente conexas) de um grafo orientado $G = (V, E)$. Digamos que $C_1, C_2, \ldots, C_k$ sejam as componentes fortes de G devolvidas nesta ordem pelo algoritmo.
- O grafo G^{CFC} é o grafo obtido de G contraindo-se cada componente forte C_i a um vértice v_i. Sabemos que $v_1, v_2, \ldots, v_k$ é uma ordenação topológica de G^{CFC} (vocês lembram, não é?).

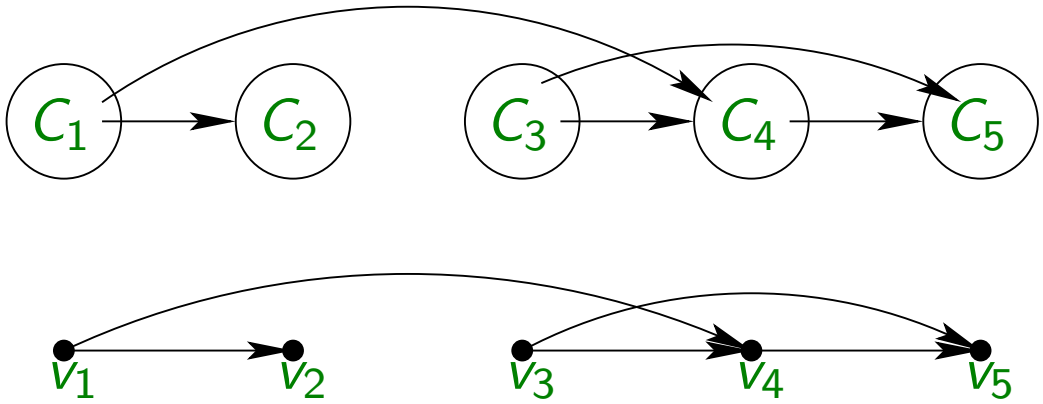
Ideia da solução geral



- Rotule cada v_i com $L'[v_i] = \min\{L(v) : v \in C_i\}$
- Resolva o problema para G^{CFC} e L'
- Copie o valor $\mathrm{MIN}'_L(v_i)$ para $\mathrm{MIN}_L[u]$ para todo $u \in C_i$

Uma (longa) pausa para meditação... Se vocês entenderam isto, resta apenas descrever como implementar esta ideia em tempo $O(V + E)$.

Construindo G^{CFC}



Podemos criar as listas de adjacências de G^{CFC} da seguinte forma:

- Criamos um vetor comp (indexado por V) tal que $\mathrm{comp}[u] = i$ se, e somente se, $u \in C_i$.
- Criamos um vetor $\mathrm{Adj}[v_1 \ldots v_k]$ de listas ligadas inicialmente vazias.
- Para cada aresta $(u, v) \in E$ sejam $i = \mathrm{comp}[u]$ e $j = \mathrm{comp}[v]$. Se $i \neq j$ então insira v_j na lista $\mathrm{Adj}[v_i]$.

Pode haver repetições na listas, mas isto não afeta o algoritmo. É possível eliminar todas as repetições em tempo $O(E)$ (Exercício).

Construindo G^{CFC}

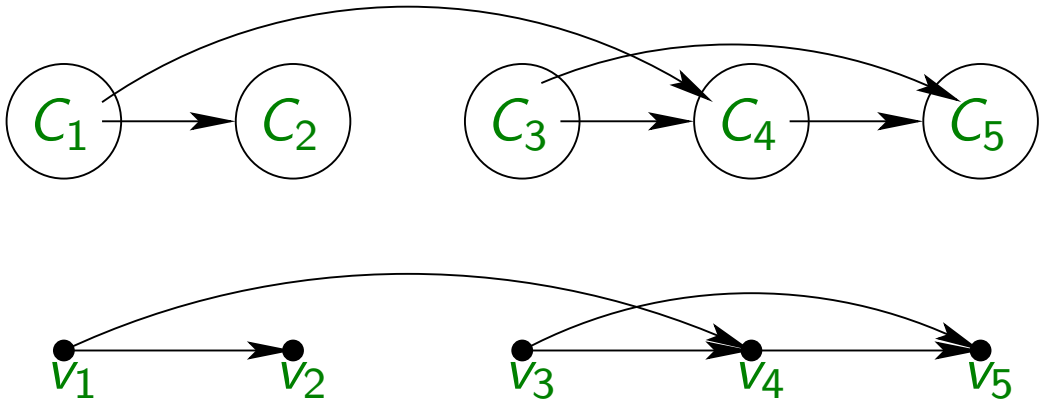
- Criamos um vetor comp (indexado por V) tal que $\text{comp}[u] = i$ se, e somente se, $u \in C_i$.
- Criamos um vetor $\text{Adj}[v_1 \ldots v_k]$ de listas ligadas inicialmente vazias.
- Para cada aresta $(u, v) \in E$ sejam $i = \text{comp}[u]$ e $j = \text{comp}[v]$. Se $i \neq j$ então insira v_j na lista $\text{Adj}[v_i]$.

Análise de complexidade:

- Linha 1 consome tempo $O(V)$ já que o algoritmo que determina as componentes fortes já devolve os C_i na ordem que queremos.
- Linha 2 consome tempo $O(k) = O(V)$.
- Linha 3 consome tempo $O(E)$.

Complexidade total: $O(V + E)$.

Solução do caso geral



1 **para** $i \leftarrow 1$ **até** k **faça** calcule $\min\{L(v) : v \in C_i\}$
2 **para** $i \leftarrow 1$ **até** k **faça** $L'[v_i] = \min\{L(v) : v \in C_i\}$
3 $\text{MIN}'_L \leftarrow \text{CasoAcíclico}(G^{\text{CFC}}, L')$
4 **para cada** C_i **faça**
5 **para cada** $u \in C_i$ **faça**
6 $\text{MIN}_L[u] \leftarrow \text{MIN}_{L'}[v_i]$

Solução do caso geral

1 **para** $i \leftarrow 1$ **até** k **faça** calcule $\min\{L(v) : v \in C_i\}$
2 **para** $i \leftarrow 1$ **até** k **faça** $L'[v_i] = \min\{L(v) : v \in C_i\}$
3 $\text{MIN}'_L \leftarrow \text{CasoAcíclico}(G^{\text{CFC}}, L')$
4 **para cada** C_i **faça**
5 **para cada** $u \in C_i$ **faça**
6 $\text{MIN}_L[u] \leftarrow \text{MIN}_{L'}[v_i]$

Análise de complexidade:

- Linha 1 consome tempo $O(|C_i|)$ para cada $i = 1, 2, \ldots, k$. No total $O(|C_1| + \cdots + |C_k|) = O(V)$.
- Linha 2 consome tempo $O(k) = O(V)$.
- Linha 3 consome tempo $O(V + E)$ pois $|V[G^{\text{CFC}}]| = k = O(V)$ e $|E[G^{\text{CFC}}]| = O(E)$.
- As linhas 4–6 consomem tempo $O(V + E)$.

Problema 2

Exercício (CLRS 23.2-5 parcial). Seja (G, ω) uma instância do problema da Árvore Geradora Mínima (AGM). Suponha que todos os pesos das arestas sejam inteiros dentro do intervalo de 1 a W onde W é uma constante. Como podemos acelerar o algoritmo de Prim?

Sugestão. A complexidade do algoritmo de Prim depende da implementação da fila de prioridades. Mostre como implementar a fila de prioridade de modo que cada operação da fila tenha custo $O(1)$!

Observação. Na questão original também se pergunta o que fazer no caso em os pesos das arestas sejam inteiros dentro do intervalo de 1 a $|V|$. Neste caso, o algoritmo de Prim pode ser acelerado usando uma estrutura de dados chamada van Emde Boas Tree que não vimos em aula. Veja o CLRS para maiores detalhes desta ED.

Solução

- Lembre que no algoritmo de Prim, a chave $d[v]$ de um vértice v fora da (sub-)árvore encontrada até então é o peso da menor aresta que liga v a algum vértice da árvore.
- Logo, as chaves pertencem a $\{1, \ldots, W\} \cup \infty$.
- Temos um vetor $L[1 \ldots W + 1]$ ($W + 1$ representa $+\infty$). Na posição $L[k]$ temos uma lista duplamente ligada onde estão armazenados os vértices cuja chave é igual a k.
- INSERT consome tempo $O(1)$ pois simplesmente inserimos o vértice com chave k em $L[k]$.

Solução

- Extract-Min consome tempo $O(1)$ pois simplesmente percorremos $O(W) = O(1)$ posições em L até encontrarmos uma lista não-vazia.
- Decrease-Key consome tempo $O(1)$. Para reduzir a chave k para um valor k' de um vértice v, remova v de $L[k]$ e o insira em $L[k']$. Note que é necessário ter para cada vértice um apontador para seu nó correspondente. Como cada lista é duplamente ligada, podemos remover um nó em tempo $O(1)$.

Tempo total: (visto em aula)
$O(V)$ Insert $+ O(V)$ Extract-Min $+ O(E)$ Decrease-Key $=$
$O(V + V + E) = O(E)$

Problema 3

Exercício (CLRS 23.2-4). Seja (G, ω) uma instância do problema da Árvore Geradora Mínima (AGM).

(a) Suponha que todos os pesos das arestas sejam inteiros dentro do intervalo de 1 a $|V|$. Como podemos acelerar o algoritmo de Kruskal?

(b) E se todos os pesos das arestas fossem inteiros dentro do intervalo de 1 a W onde W é uma constante.

Sugestão. Qual é o passo mais custoso no algoritmo de Kruskal?

Solução

Túnel do tempo. O algoritmo COUNTING-SORT ordena um vetor $A[1 \ldots n]$ em que cada $A[i]$ é um inteiro em $[0 \ldots k]$ em tempo $O(n + k)$.

(a) Suponha que todos os pesos das arestas sejam inteiros dentro do intervalo de 1 a $|V|$. Assim, podemos ordenar as arestas em ordem crescente de peso em tempo $O(E + V) = O(E)$.

Os demais passos do algoritmo Kruskal consomem tempo $O(E\alpha(V))$ (usando *disjoint sets forest*).

Logo, o tempo total é $O(E + E\alpha(V)) = O(E\alpha(V))$.

Solução

(b) Suponha que todos os pesos das arestas sejam inteiros dentro do intervalo de 1 a W onde W é uma constante.

Novamente podemos usar o COUNTING-SORT para ordenar as arestas em ordem crescente de peso em tempo $O(E + W) = O(E)$. Mesmo W sendo uma constante, não é possível fazer melhor (a cota inferior é $\Omega(E)$).

Assim, o tempo total é $O(E + E\alpha(V)) = O(E\alpha(V))$.