Grafos – Busca em largura

A busca em largura é um dos algoritmos mais simples para se pesquisar um grafo e é arquétipo de muitos algoritmos de grafos importantes.

O algoritmo de caminhos mais curtos de origem única de Dijkstra e o algoritmo de árvore espalhada mínima de Prim utilizam idéias semelhantes às que aparecem na busca em largura.

A busca em largura baseia-se em partindo de um determinado vértice ***s*** (origem) explorar sistematicamente as arestas do grafo determinando cada vértice acessível a partir de ***s***.

UNIVASF

Grafos – Busca em largura

O algoritmo calcula a distância (menor número de arestas) desde ***s*** até todos os vértices acessíveis partindo do mesmo.

Ele também produz uma "árvore primeiro na extensão" com raiz ***s*** que contém todos os vértices acessíveis. Para qualquer vértice acessível ***v*** a partir de ***s***, o caminho na árvore primeiro na extensão de ***s*** até ***v*** corresponde a um "caminho mais curto" de ***s*** até ***v*** em **G** (grafo), ou seja, um caminho que contém o número mínimo de arestas.

O algoritmo funciona sobre grafos orientados e também não orientados.

UNIVASF

Grafos – Busca em largura

Para controlar o andamento, a busca em largura pinta cada vértice de branco, cinza ou preto.

No início, todos os vértices são brancos, e mais tarde eles podem se tornar acinzentadas e depois pretos.

Um vértice é descoberto na primeira vez em que é encontrado durante a pesquisa, e nesse momento ele se torna não branco. Portanto, vértices em cor cinza e preta foram descobertos, mas a busca em largura faz distinção entre eles para assegurar que a pesquisa continuará de maneira a seguir primeiro na extensão.

UNIVASF

Grafos – Busca em largura

Considerando um grafo **G** = (**V**, **E**). Se (**u**, **v**) $\in$ **E** e o vértice ***u*** é preto, então o vértice ***v*** é cinza ou preto; isto é, todos os vértices adjacentes a vértices pretos foram descobertos. Vértices de cor cinza podem ter alguns vértices adjacentes brancos; eles representam a fronteira entre vértices descobertos e não descobertos.

Sendo assim, a busca em largura constrói uma árvore primeiro na extensão, contendo inicialmente apenas sua raiz, a qual é o vértice de origem ***s***. Sempre que um vértice branco ***v*** é descoberto no curso da varredura da lista de adjacências de um vértice ***u*** já descoberto, o vértice ***v*** e a aresta (***u***, ***v***) são adicionados à árvore.

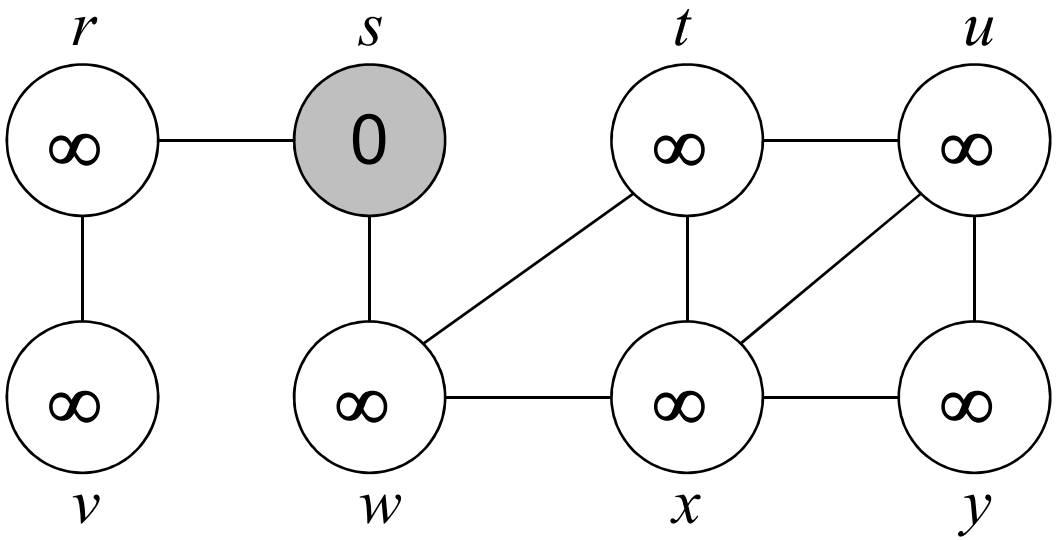
UNIVASF

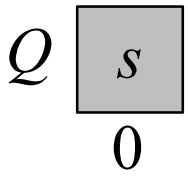
Grafos – Busca em largura

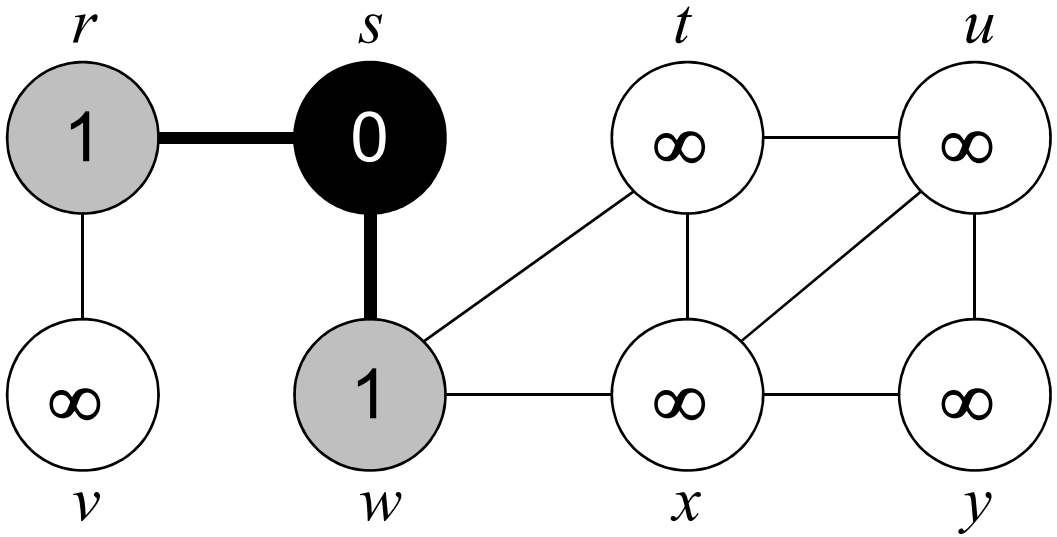
Dizemos que ***u*** é o predecessor ou pai de ***v*** na árvore primeiro na extensão. Tendo em vista que um vértice é descoberto no máximo uma vez, ele tem no máximo um pai. Relacionamentos de ancestral e descendente na árvore primeiro na extensão são definidos em relação à raiz ***s*** da maneira usual: se ***u*** está em um caminho na árvore a partir da raiz ***s*** até o vértice ***v***, então ***u*** é um ancestral de ***v***, e ***v*** é um descendente de ***u***.

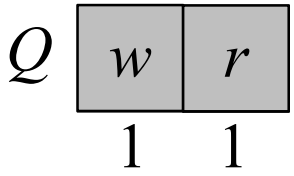
Para uma melhor compreensão vamos observar como ocorre este processo no grafo apresentado no próximo slide.

Grafos – Busca em largura

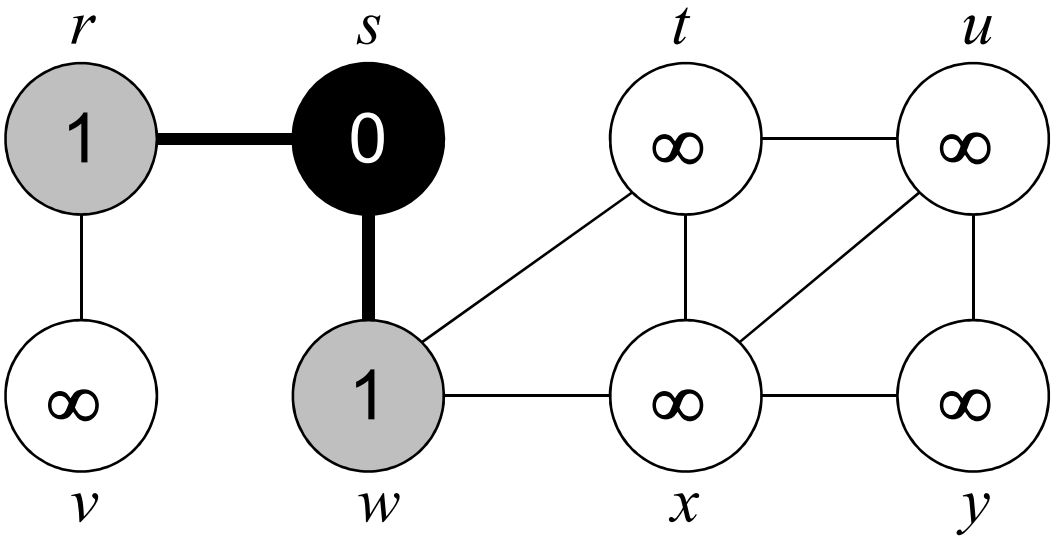


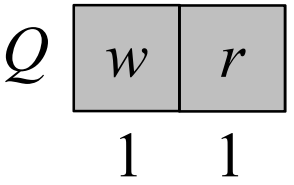


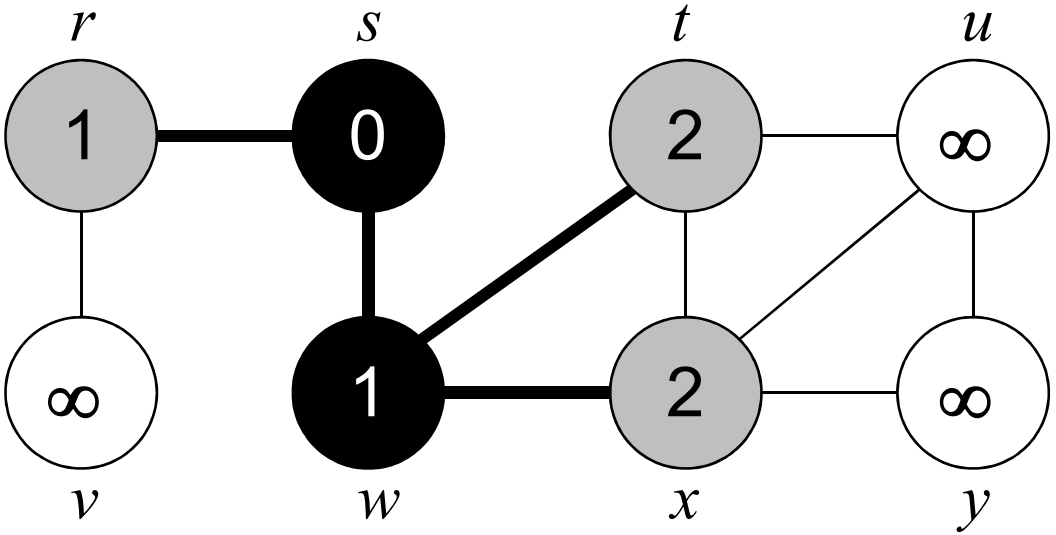


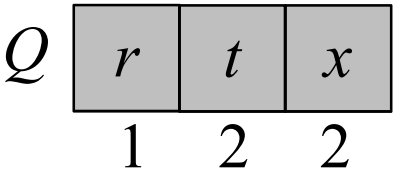


Grafos – Busca em largura

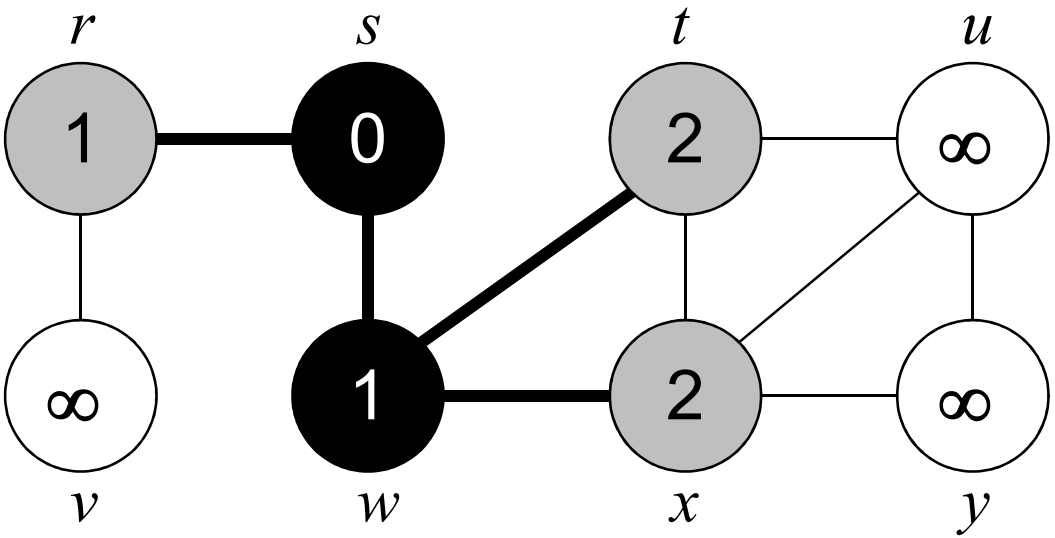


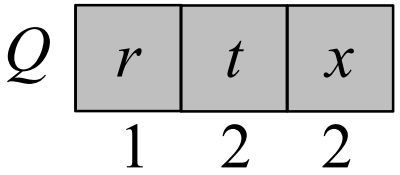


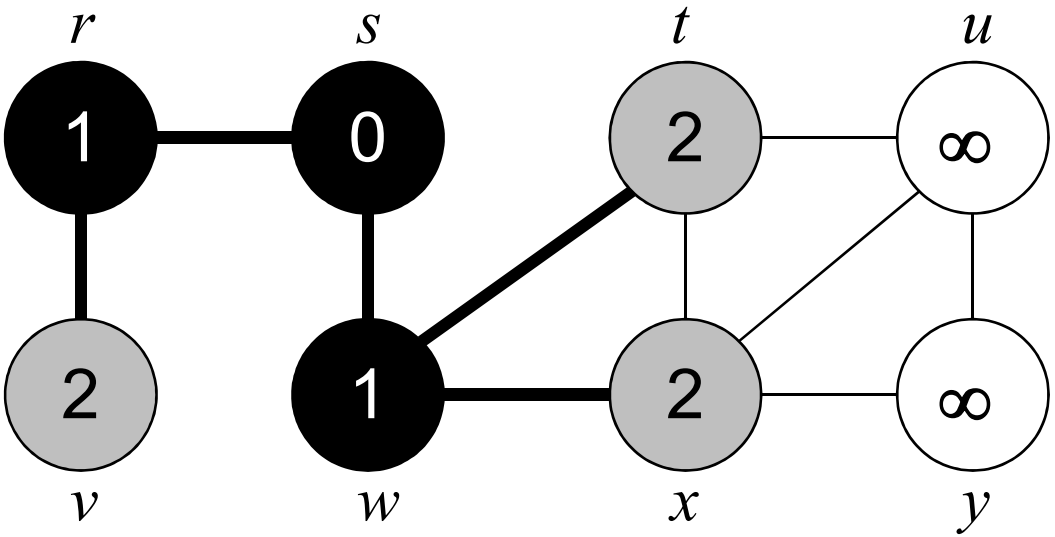


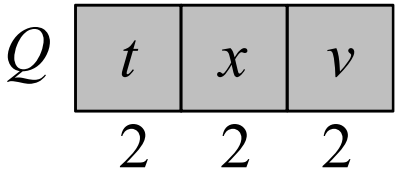


Grafos – Busca em largura

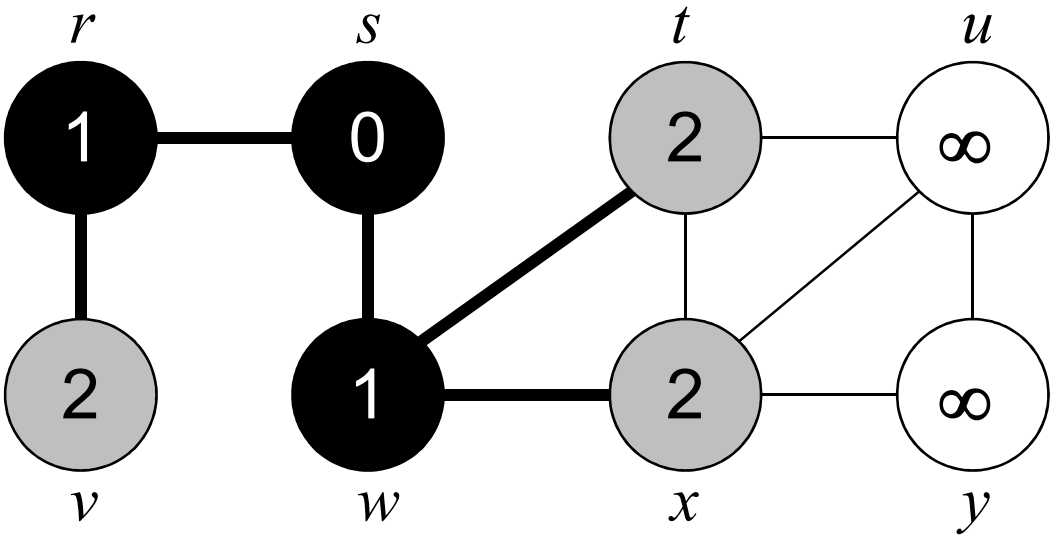


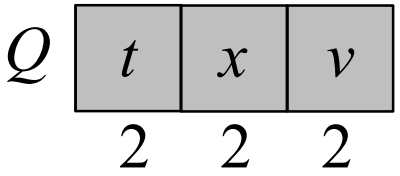


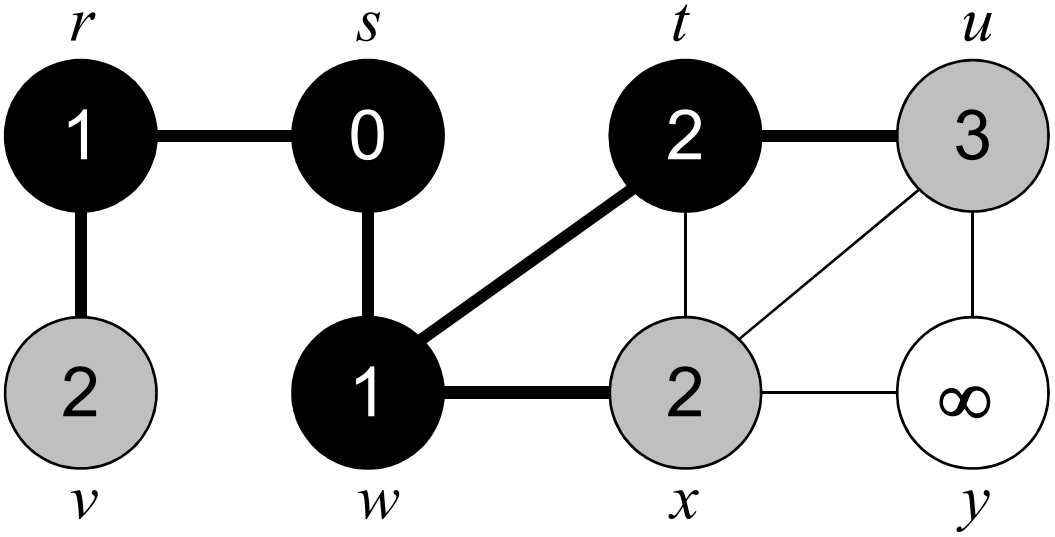


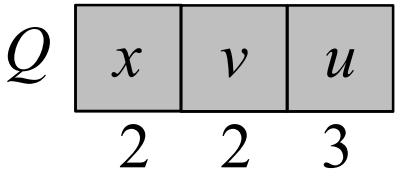


Grafos – Busca em largura

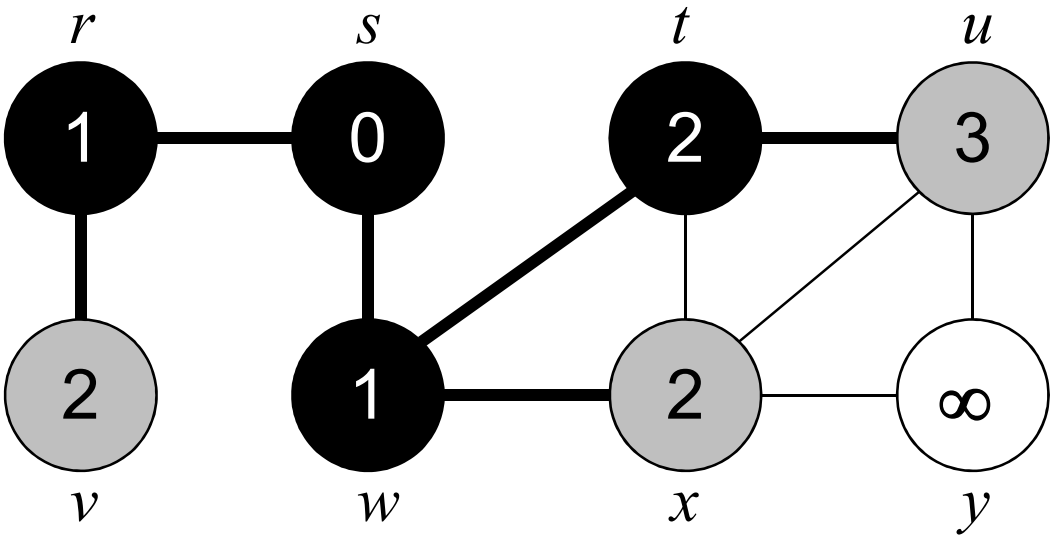


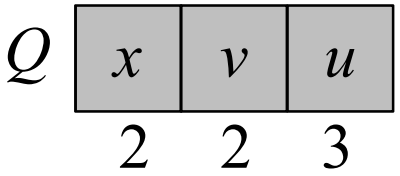


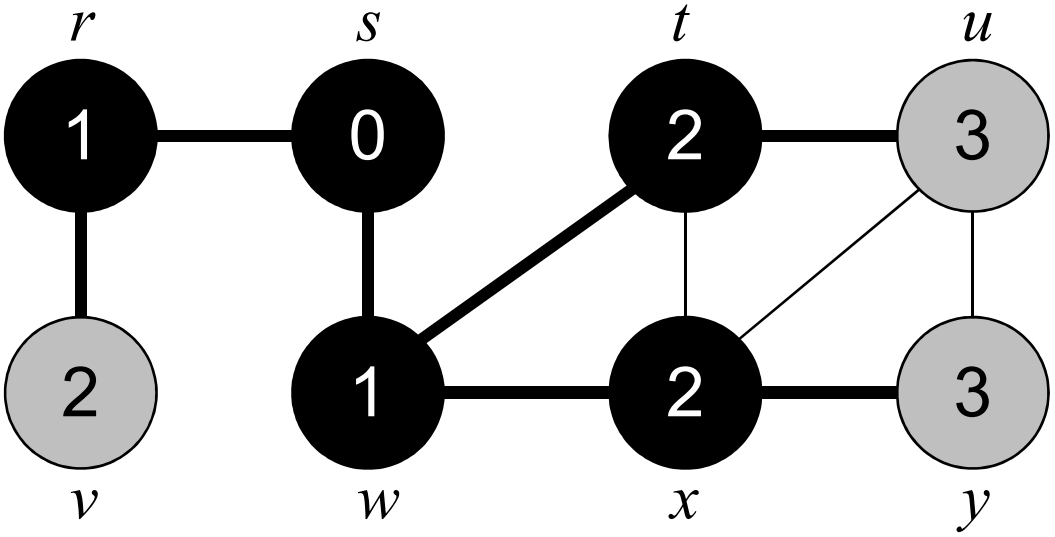


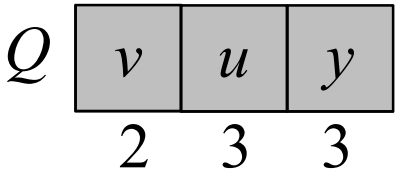


Grafos – Busca em largura

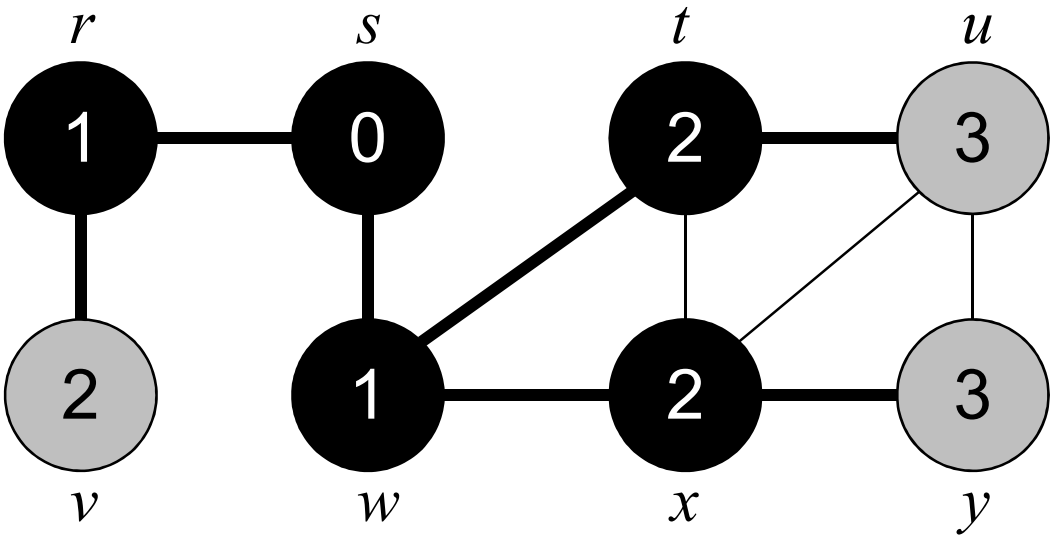


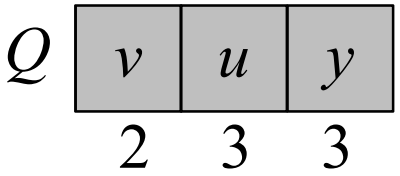


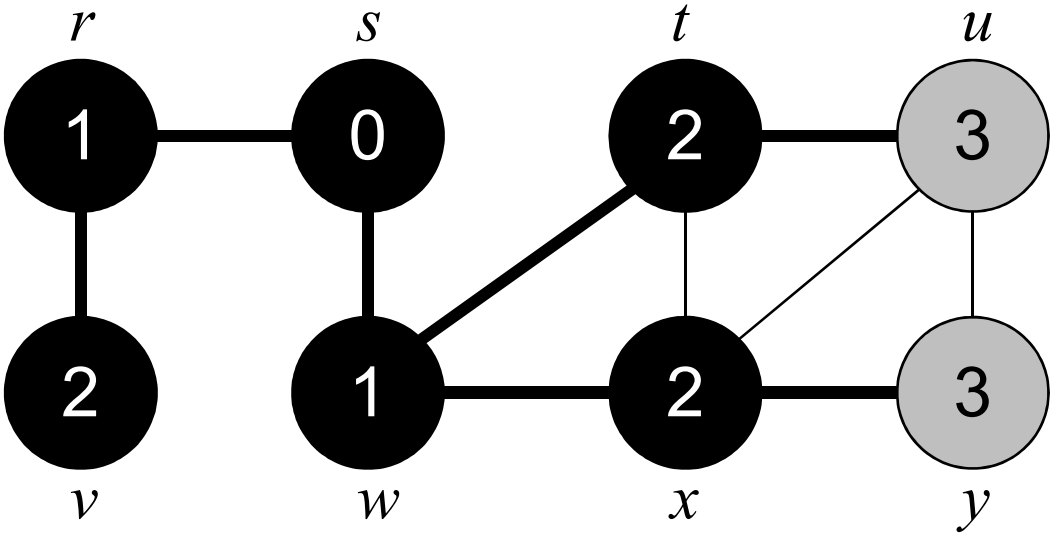


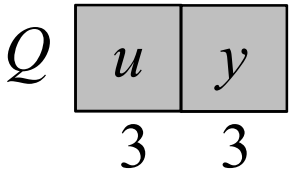


Grafos – Busca em largura

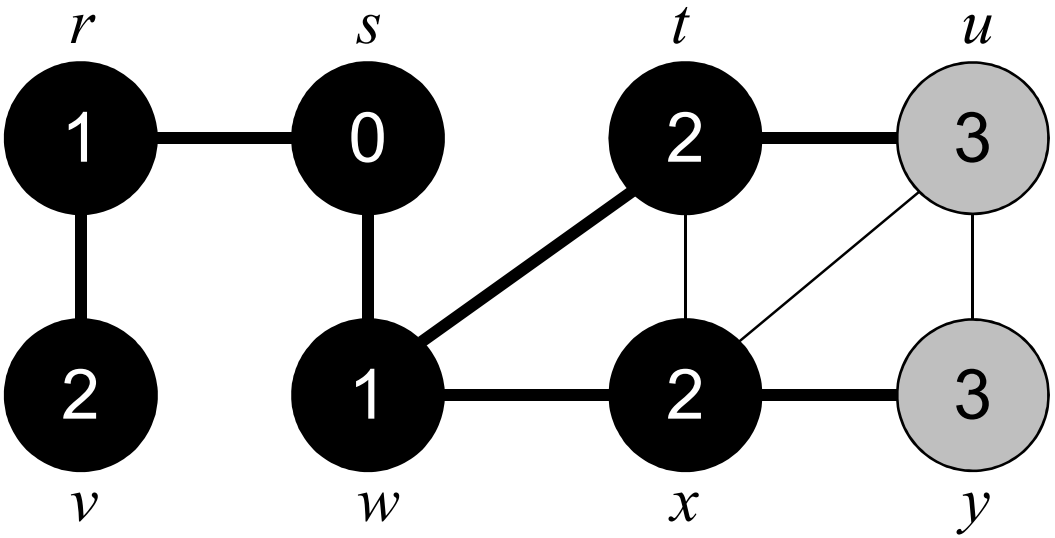


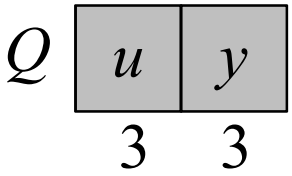


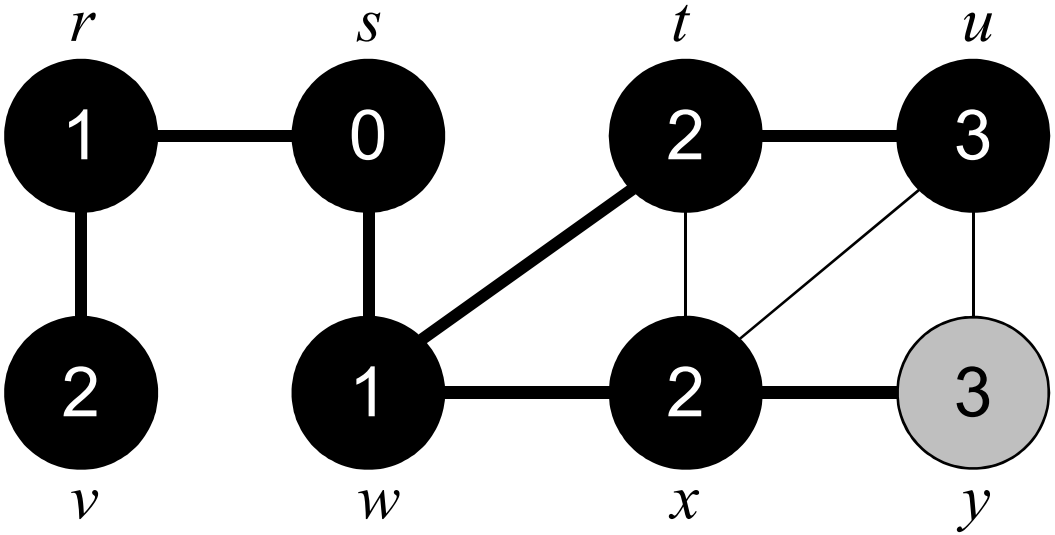


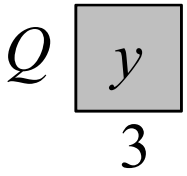


Grafos – Busca em largura

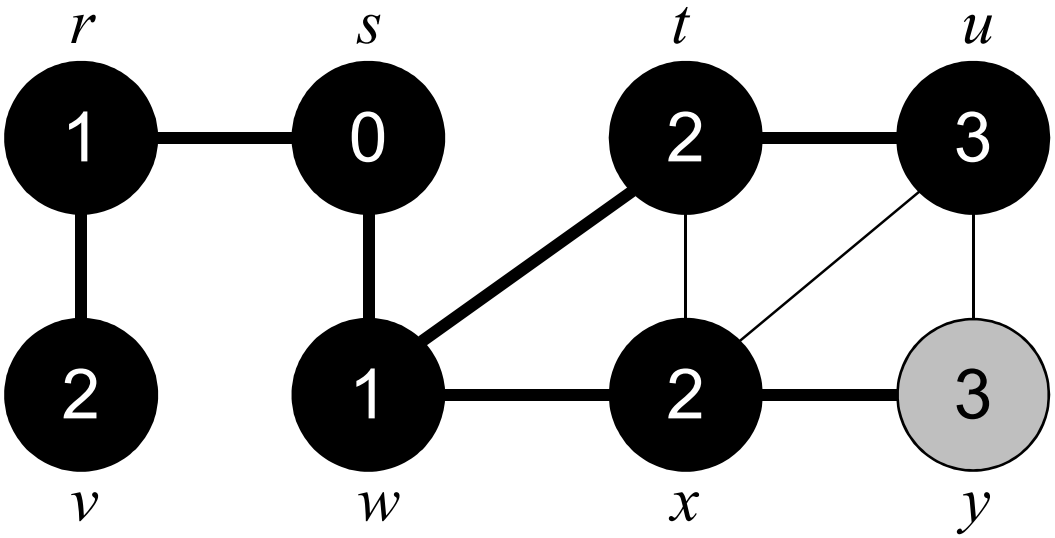




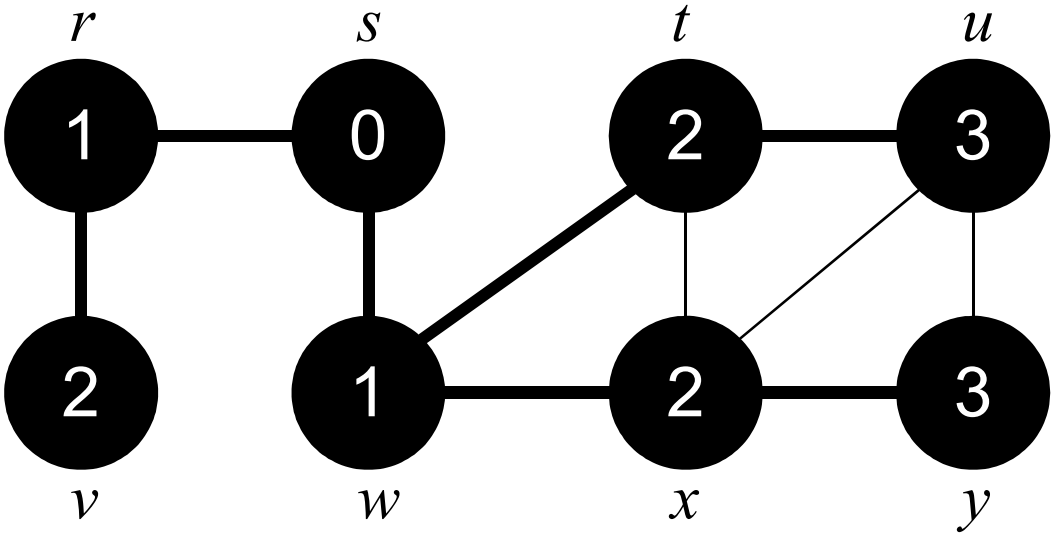




Grafos – Busca em largura



Q | y | 3



Q $\varnothing$

Grafos – Busca em largura

Com base no que foi estudado, codifique uma função na linguagem C que implemente o procedimento de busca em largura discutido, pressuponha que o grafo de entrada **G** = (**V**, **E**) é representado com o uso de listas de adjacências (representação encadeada em vetor).

Dica: Mantenha estruturas de dados adicionais para armazenar informações relativas a cada vértice no grafo. A cor de cada vértice $u \in V$ pode ser armazenada na variável cor[***u***], e o predecessor de ***u*** pode ser armazenado na variável pai[***u***]. Se ***u*** não tem nenhum predecessor (por exemplo, se ***u*** = ***s*** ou se ***u*** não foi descoberto), então pai[***u***] = NULL. A distância desde a origem ***s*** até o vértice ***u*** calculada pelo algoritmo pode ser armazenada em d[***u***]. Para facilitar a implementação o algoritmo também pode se utilizar de uma fila ***Q*** (FIFO) para gerenciar o conjunto de vértices de cor cinza.

```
BFS(G, s) /*Procedimento apresentado no livro Algoritmos, Cormen at all*/
1   for cada vértice u ∈ V[G] – {s}
2      do cor[u] <- BRANCO
3           d[u] <- ∞
4           pai[u] <- NULL
5   cor[s] <- CINZA
6   d[s] <- 0
7   pai[s] <- NULL
8   Q <- ∅
9   ENQUEUE(Q, s) /*Insere s na fila Q*/
10 while Q ≠ ∅
11  do u <- DEQUEUE(Q) /*Consulta e remove o primeiro elemento da fila Q*/
12       for cada v <- Adj[u]
13       do if cor[v] = BRANCO
14            then cor[v] <-CINZA
15                 d[v] <- d[u] + 1
16                 pai[v] <- u
17                 ENQUEUE (Q, v)
18       cor[u] <- PRETO
```