MC-202
Algoritmos em Grafos

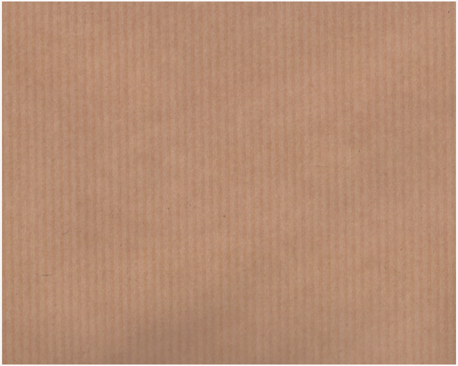
Lehilton Pedrosa
lehilton@ic.unicamp.br

Universidade Estadual de Campinas

Segundo semestre de 2024

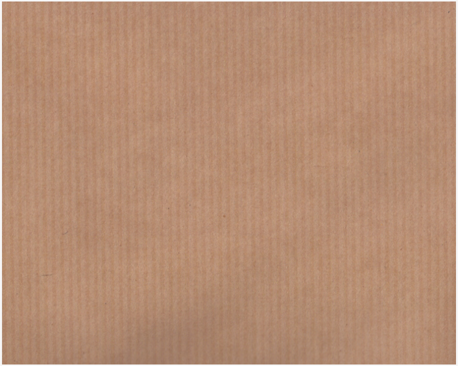
Corte de material

- uma fábrica utiliza precisa cortar papelão retangular



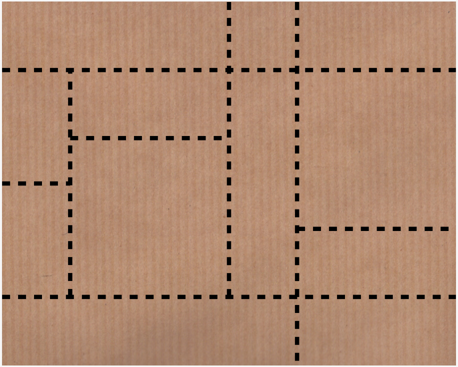
Corte de material

- uma fábrica utiliza precisa cortar papelão retangular
- ela utilizada uma grande guilhotina (maior que o papelão)



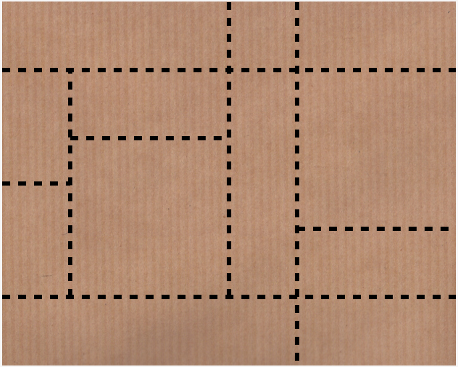
Corte de material

- uma fábrica utiliza precisa cortar papelão retangular
- ela utilizada uma grande guilhotina (maior que o papelão)
- existe um padrão de corte para evitar desperdício



Corte de material

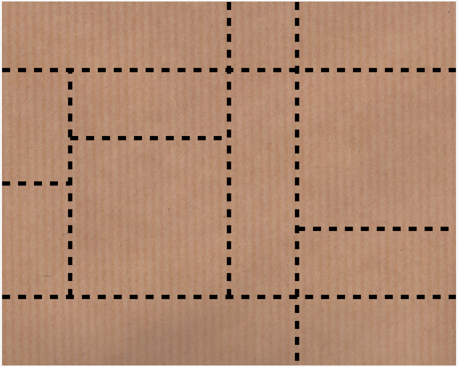
- uma fábrica utiliza precisa cortar papelão retangular
- ela utilizada uma grande guilhotina (maior que o papelão)
- existe um padrão de corte para evitar desperdício



1. o corte pode ser feito pela guilhotina?

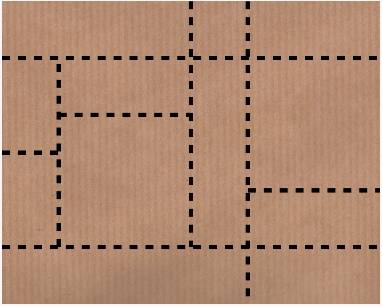
Corte de material

- uma fábrica utiliza precisa cortar papelão retangular
- ela utilizada uma grande guilhotina (maior que o papelão)
- existe um padrão de corte para evitar desperdício

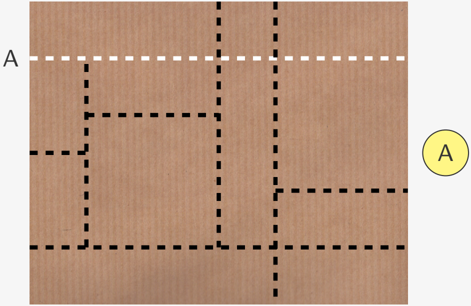


1. o corte pode ser feito pela guilhotina?
2. em que ordem eles devem ser feitos?

Modelando como um grafo

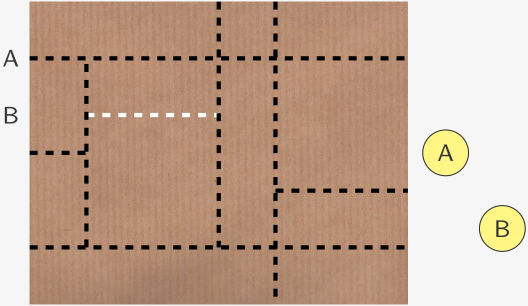


Modelando como um grafo



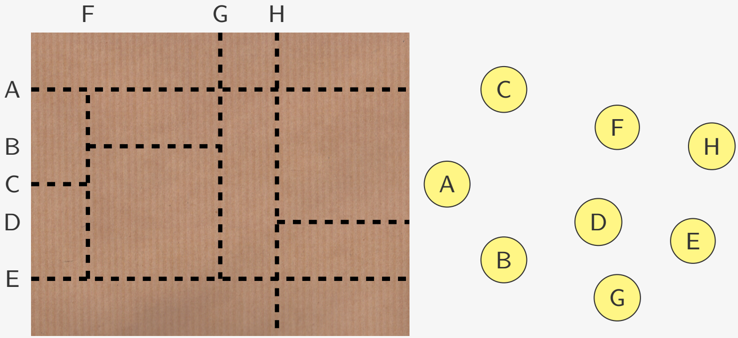
- vértices: consideramos cada corte da guilhotina

Modelando como um grafo



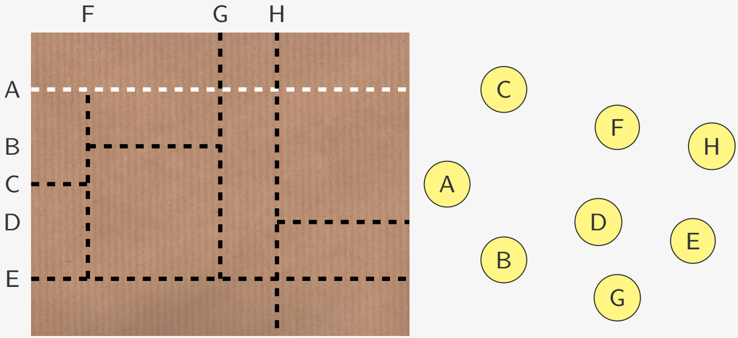
- vértices: consideramos cada corte da guilhotina

Modelando como um grafo



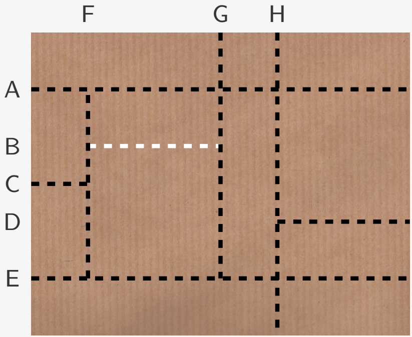
- vértices: consideramos cada corte da guilhotina

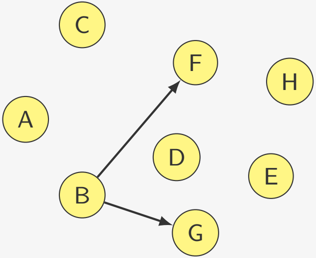
Modelando como um grafo



- vértices: consideramos cada corte da guilhotina
- arestas: se um corte depende de um corte anterior

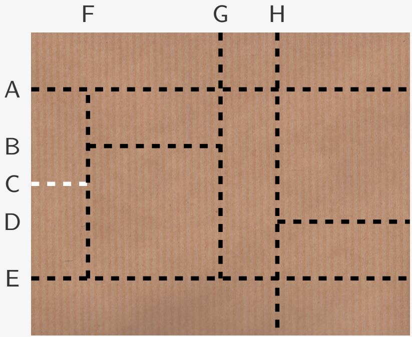
Modelando como um grafo

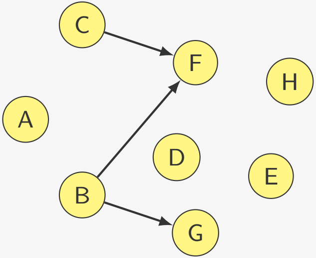




- vértices: consideramos cada corte da guilhotina
- arestas: se um corte depende de um corte anterior

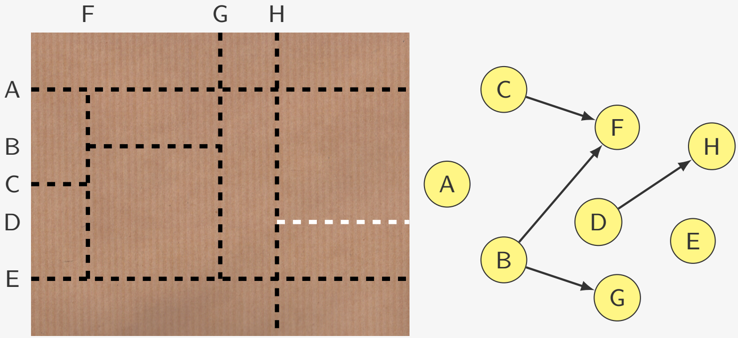
Modelando como um grafo





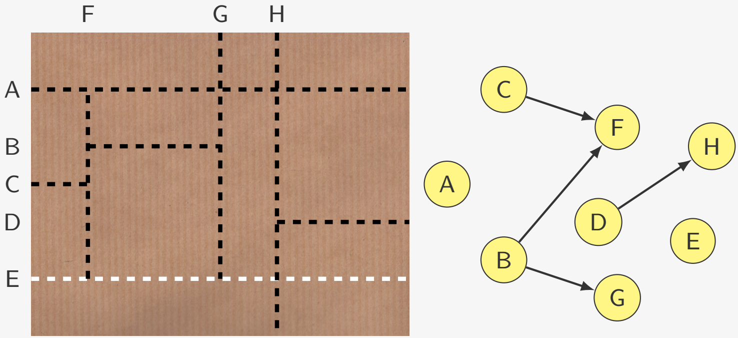
- vértices: consideramos cada corte da guilhotina
- arestas: se um corte depende de um corte anterior

Modelando como um grafo



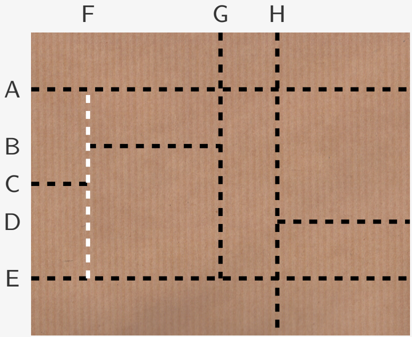
- vértices: consideramos cada corte da guilhotina
- arestas: se um corte depende de um corte anterior

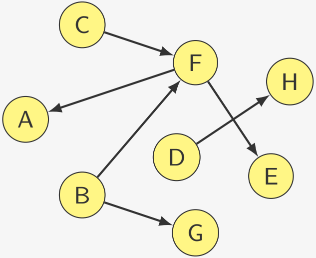
Modelando como um grafo



- vértices: consideramos cada corte da guilhotina
- arestas: se um corte depende de um corte anterior

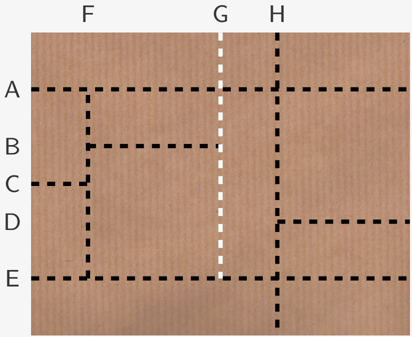
Modelando como um grafo

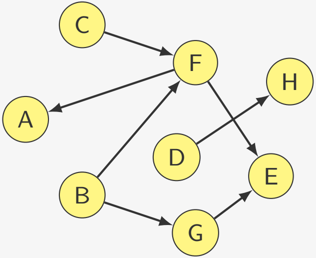




- vértices: consideramos cada corte da guilhotina
- arestas: se um corte depende de um corte anterior

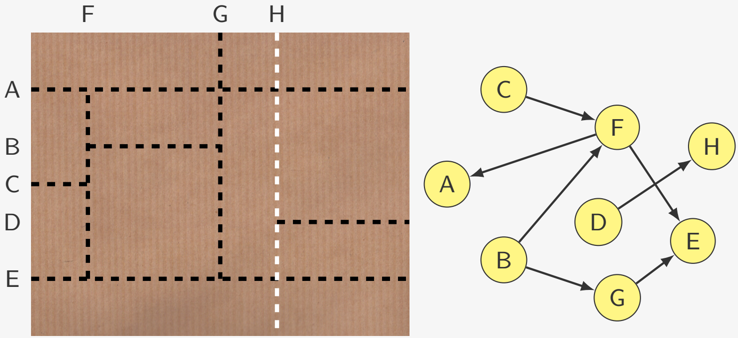
Modelando como um grafo





- vértices: consideramos cada corte da guilhotina
- arestas: se um corte depende de um corte anterior

Modelando como um grafo



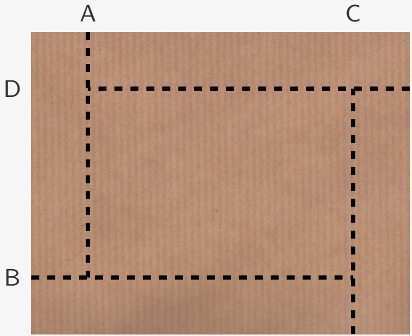
- vértices: consideramos cada corte da guilhotina
- arestas: se um corte depende de um corte anterior

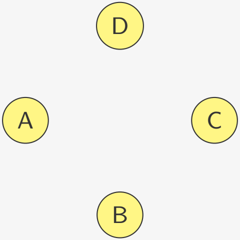
Viabilidade do padrão

Sempre é possível cortar um padrão com a guilhotina?

Viabilidade do padrão

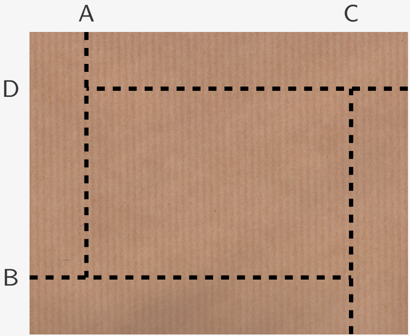
Sempre é possível cortar um padrão com a guilhotina?

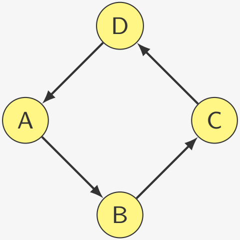




Viabilidade do padrão

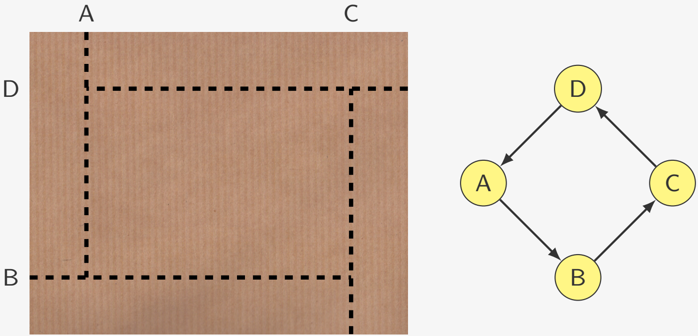
Sempre é possível cortar um padrão com a guilhotina?





Viabilidade do padrão

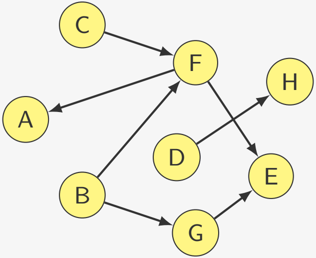
Sempre é possível cortar um padrão com a guilhotina?



Conclusão: Um padrão de corte é viável se, e somente se, o grafo dos cortes for acíclico.

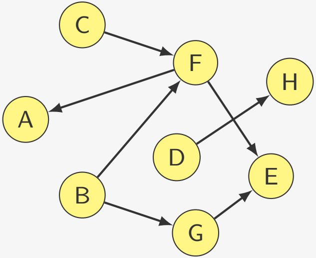
Ordenação topológica

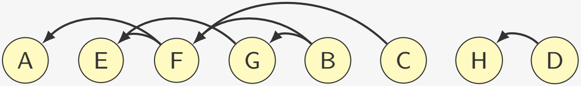
Uma ordenação topológica de um grafo acíclico é uma ordenação dos vértices cujas arestas estão na mesma direção.



Ordenação topológica

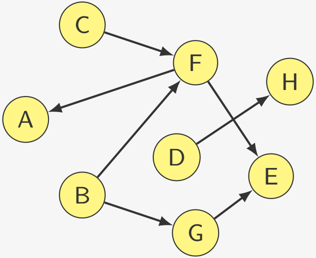
Uma ordenação topológica de um grafo acíclico é uma ordenação dos vértices cujas arestas estão na mesma direção.

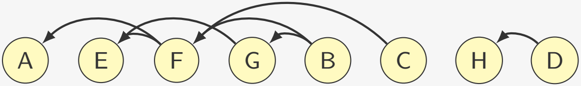




Ordenação topológica

Uma ordenação topológica de um grafo acíclico é uma ordenação dos vértices cujas arestas estão na mesma direção.

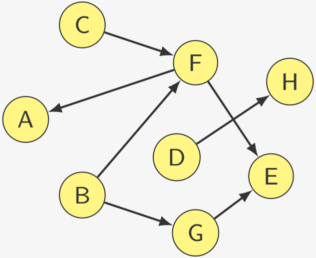


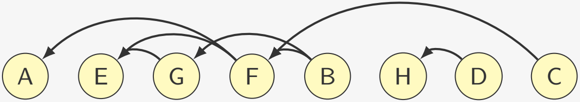


Observe que pode haver várias ordenações topológicas.

Ordenação topológica

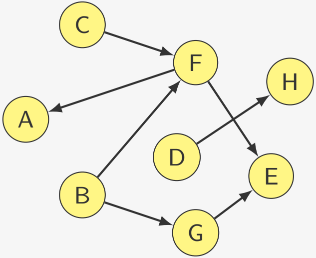
Uma ordenação topológica de um grafo acíclico é uma ordenação dos vértices cujas arestas estão na mesma direção.



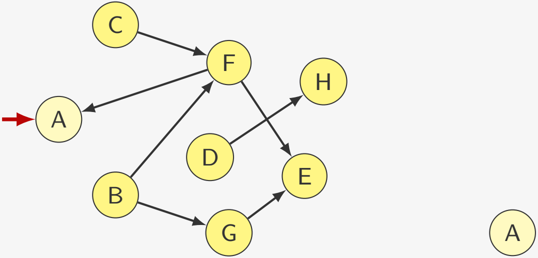


Observe que pode haver várias ordenações topológicas.

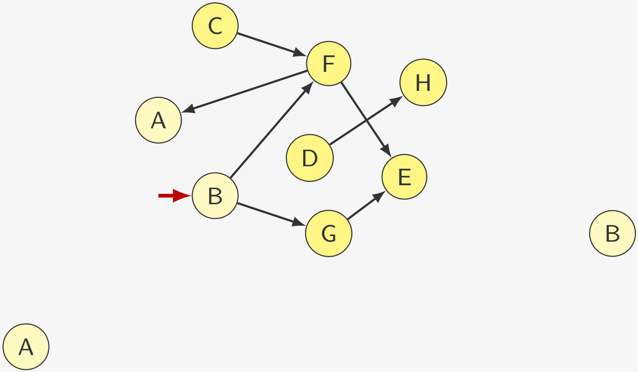
Encontrando uma ordem de corte



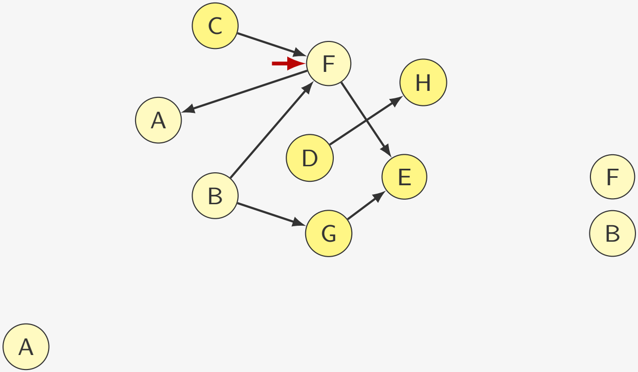
Encontrando uma ordem de corte



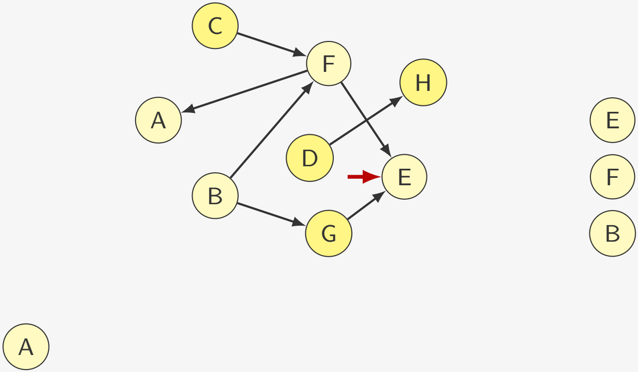
Encontrando uma ordem de corte



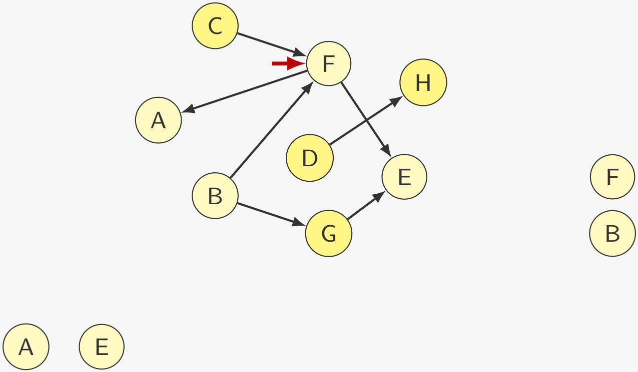
Encontrando uma ordem de corte



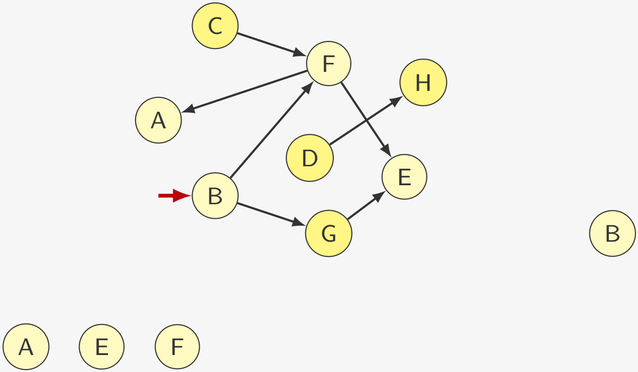
Encontrando uma ordem de corte



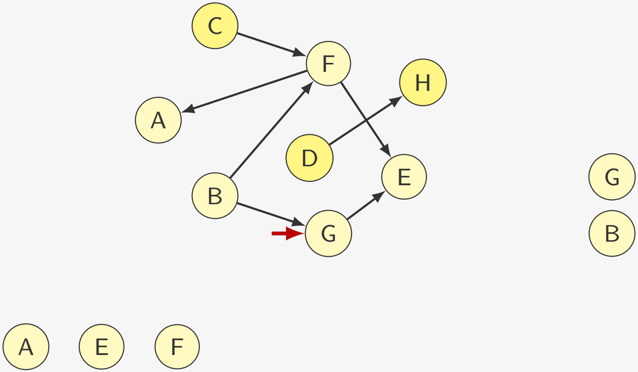
Encontrando uma ordem de corte



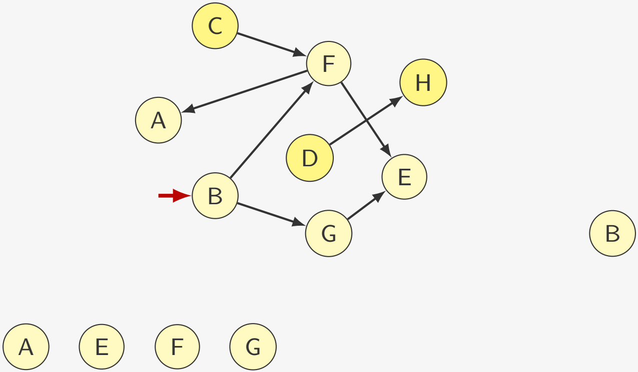
Encontrando uma ordem de corte



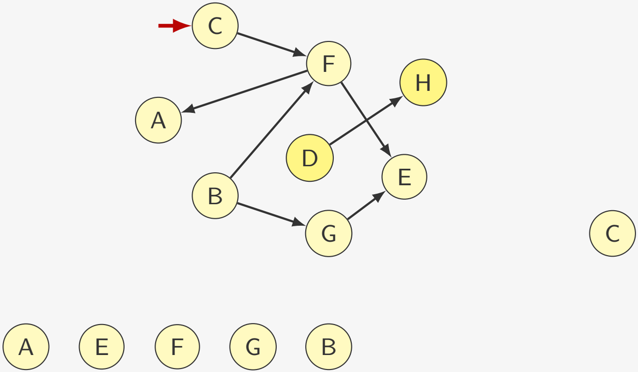
Encontrando uma ordem de corte



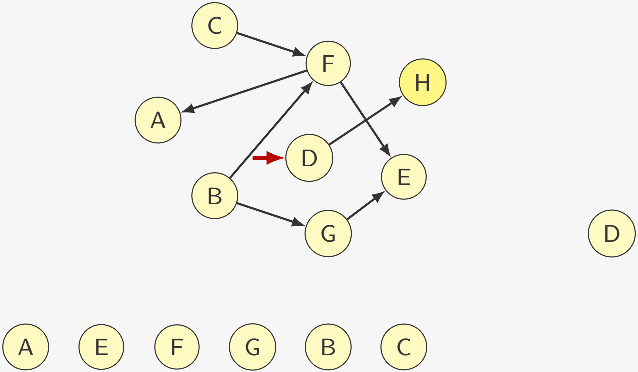
Encontrando uma ordem de corte



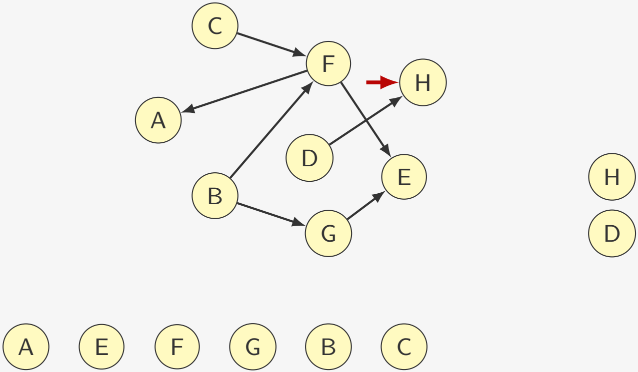
Encontrando uma ordem de corte



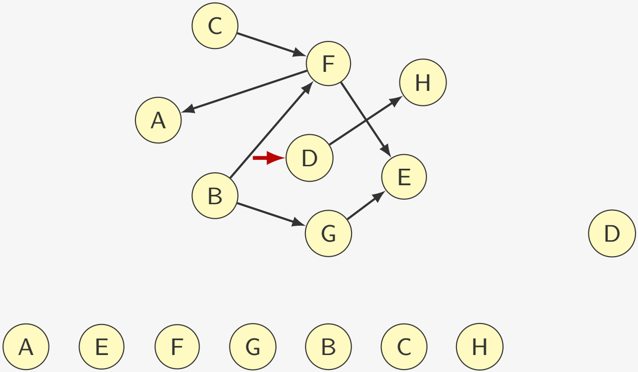
Encontrando uma ordem de corte



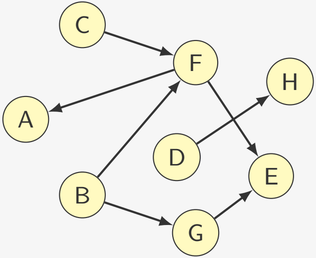
Encontrando uma ordem de corte



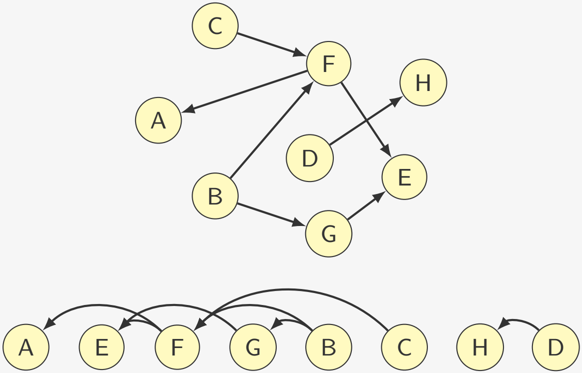
Encontrando uma ordem de corte



Encontrando uma ordem de corte



Encontrando uma ordem de corte



Como encontrar um ordenação topológica?

Considere um vértice u do DAG:

Como encontrar um ordenação topológica?

Considere um vértice u do DAG:

- Todo v tal que (u, v) é arco deve aparecer antes de u

Como encontrar um ordenação topológica?

Considere um vértice u do DAG:

- Todo v tal que (u, v) é arco deve aparecer antes de u
- Todo w tal que (v, w) e arco deve aparecer antes de v

Como encontrar um ordenação topológica?

Considere um vértice u do DAG:

- Todo v tal que (u, v) é arco deve aparecer antes de u
- Todo w tal que (v, w) e arco deve aparecer antes de v
- E assim por diante

Como encontrar um ordenação topológica?

Considere um vértice u do DAG:

- Todo v tal que (u, v) é arco deve aparecer antes de u
- Todo w tal que (v, w) e arco deve aparecer antes de v
- E assim por diante

Devemos considerar todos w para os quais existe caminho de u para w antes de considerar u

Como encontrar um ordenação topológica?

Considere um vértice u do DAG:

- Todo v tal que (u, v) é arco deve aparecer antes de u
- Todo w tal que (v, w) e arco deve aparecer antes de v
- E assim por diante

Devemos considerar todos w para os quais existe caminho de u para w antes de considerar u

- Lembra uma pós-ordem em árvores binárias...

Como encontrar um ordenação topológica?

Considere um vértice u do DAG:

- Todo v tal que (u, v) é arco deve aparecer antes de u
- Todo w tal que (v, w) e arco deve aparecer antes de v
- E assim por diante

Devemos considerar todos w para os quais existe caminho de u para w antes de considerar u

- Lembra uma pós-ordem em árvores binárias...

Como encontrar todo w tal que existe caminho de u para w?

Como encontrar um ordenação topológica?

Considere um vértice u do DAG:

- Todo v tal que (u, v) é arco deve aparecer antes de u
- Todo w tal que (v, w) e arco deve aparecer antes de v
- E assim por diante

Devemos considerar todos w para os quais existe caminho de u para w antes de considerar u

- Lembra uma pós-ordem em árvores binárias...

Como encontrar todo w tal que existe caminho de u para w?

- Busca em profundidade

Implementação

```
void ordenacao_topologica(p_grafo g) {
```

Implementação

```
void ordenacao_topologica(p_grafo g) {
  int s, *visitado = malloc(g->n * sizeof(int));
```

Implementação

```
void ordenacao_topologica(p_grafo g) {
  int s, *visitado = malloc(g->n * sizeof(int));
  for (s = 0; s < g->n; s++)
    visitado[s] = 0;
```

Implementação

```c
void ordenacao_topologica(p_grafo g) {
  int s, *visitado = malloc(g->n * sizeof(int));
  for (s = 0; s < g->n; s++)
    visitado[s] = 0;
  for (s = 0; s < g->n; s++)
    if (!visitado[s])
```

Implementação

```c
void ordenacao_topologica(p_grafo g) {
  int s, *visitado = malloc(g->n * sizeof(int));
  for (s = 0; s < g->n; s++)
    visitado[s] = 0;
  for (s = 0; s < g->n; s++)
    if (!visitado[s])
      visita_rec(g, visitado, s);
```

Implementação

```c
void ordenacao_topologica(p_grafo g) {
  int s, *visitado = malloc(g->n * sizeof(int));
  for (s = 0; s < g->n; s++)
    visitado[s] = 0;
  for (s = 0; s < g->n; s++)
    if (!visitado[s])
      visita_rec(g, visitado, s);
  free(visitado);
```

Implementação

```
void ordenacao_topologica(p_grafo g) {
  int s, *visitado = malloc(g->n * sizeof(int));
  for (s = 0; s < g->n; s++)
    visitado[s] = 0;
  for (s = 0; s < g->n; s++)
    if (!visitado[s])
      visita_rec(g, visitado, s);
  free(visitado);
  printf("\n");
}
```

Implementação

```
void ordenacao_topologica(p_grafo g) {
  int s, *visitado = malloc(g->n * sizeof(int));
  for (s = 0; s < g->n; s++)
    visitado[s] = 0;
  for (s = 0; s < g->n; s++)
    if (!visitado[s])
      visita_rec(g, visitado, s);
  free(visitado);
  printf("\n");
}
```

```
void visita_rec(p_grafo g, int *visitado, int v) {
```

Implementação

```c
void ordenacao_topologica(p_grafo g) {
  int s, *visitado = malloc(g->n * sizeof(int));
  for (s = 0; s < g->n; s++)
    visitado[s] = 0;
  for (s = 0; s < g->n; s++)
    if (!visitado[s])
      visita_rec(g, visitado, s);
  free(visitado);
  printf("\n");
}
```

```c
void visita_rec(p_grafo g, int *visitado, int v) {
  p_no t;
  visitado[v] = 1;
```

Implementação

```c
void ordenacao_topologica(p_grafo g) {
  int s, *visitado = malloc(g->n * sizeof(int));
  for (s = 0; s < g->n; s++)
    visitado[s] = 0;
  for (s = 0; s < g->n; s++)
    if (!visitado[s])
      visita_rec(g, visitado, s);
  free(visitado);
  printf("\n");
}
```

```c
void visita_rec(p_grafo g, int *visitado, int v) {
  p_no t;
  visitado[v] = 1;
  for (t = g->adj[v]; t != NULL; t = t->prox)
```

Implementação

```c
void ordenacao_topologica(p_grafo g) {
  int s, *visitado = malloc(g->n * sizeof(int));
  for (s = 0; s < g->n; s++)
    visitado[s] = 0;
  for (s = 0; s < g->n; s++)
    if (!visitado[s])
      visita_rec(g, visitado, s);
  free(visitado);
  printf("\n");
}
```

```c
void visita_rec(p_grafo g, int *visitado, int v) {
  p_no t;
  visitado[v] = 1;
  for (t = g->adj[v]; t != NULL; t = t->prox)
    if (!visitado[t->v])
```

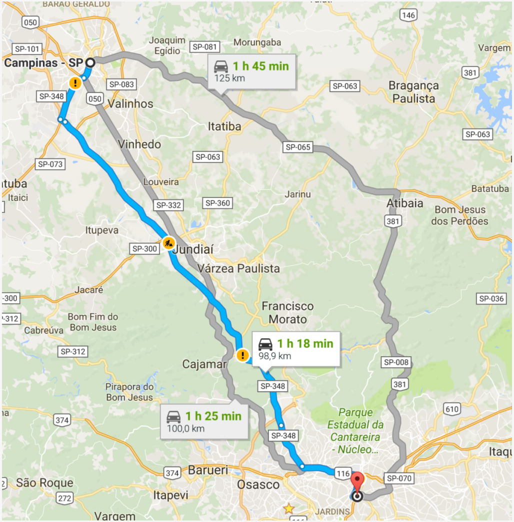
Implementação

```
void ordenacao_topologica(p_grafo g) {
  int s, *visitado = malloc(g->n * sizeof(int));
  for (s = 0; s < g->n; s++)
    visitado[s] = 0;
  for (s = 0; s < g->n; s++)
    if (!visitado[s])
      visita_rec(g, visitado, s);
  free(visitado);
  printf("\n");
}
```

```
void visita_rec(p_grafo g, int *visitado, int v) {
  p_no t;
  visitado[v] = 1;
  for (t = g->adj[v]; t != NULL; t = t->prox)
    if (!visitado[t->v])
      visita_rec(g, visitado, t->v);
  printf("%d ", v);
}
```

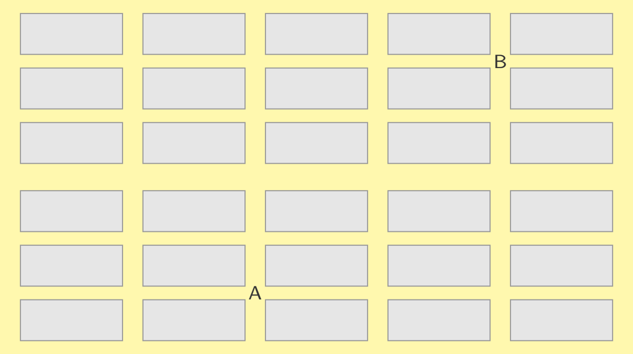
Encontrando o menor caminho

Como encontrar o menor tempo para ir de A para B?



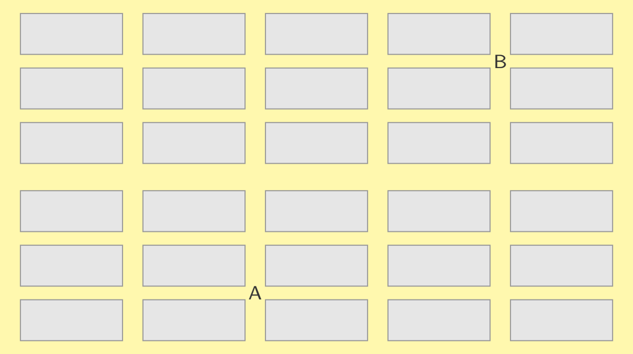
Encontrando o menor caminho

Como encontrar o menor tempo para ir de A para B?



Encontrando o menor caminho

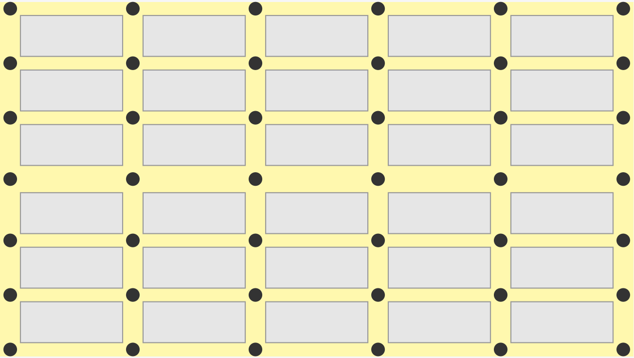
Como encontrar o menor tempo para ir de A para B?



Modelamos como um digrafo com pesos nos arcos:

Encontrando o menor caminho

Como encontrar o menor tempo para ir de A para B?

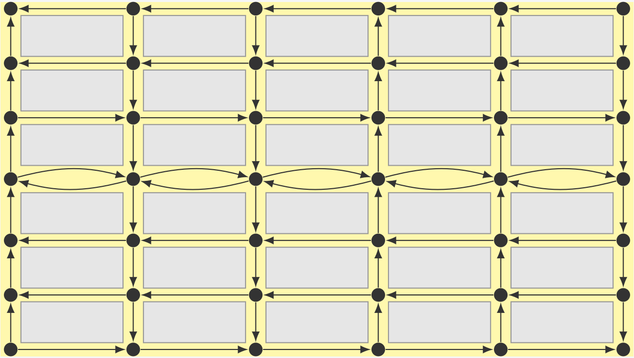


Modelamos como um digrafo com pesos nos arcos:

- Um vértice em cada cruzamento

Encontrando o menor caminho

Como encontrar o menor tempo para ir de A para B?

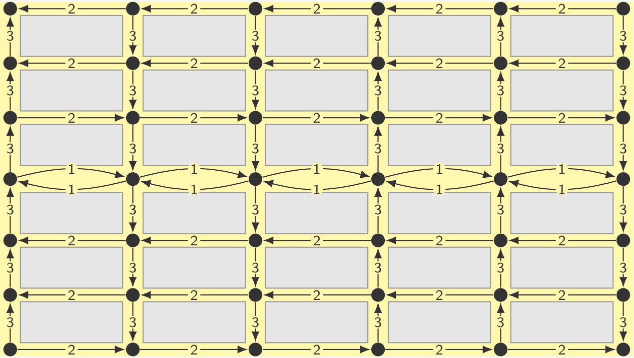


Modelamos como um digrafo com pesos nos arcos:

- Um vértice em cada cruzamento
- Um arco entre vértices consecutivos

Encontrando o menor caminho

Como encontrar o menor tempo para ir de A para B?

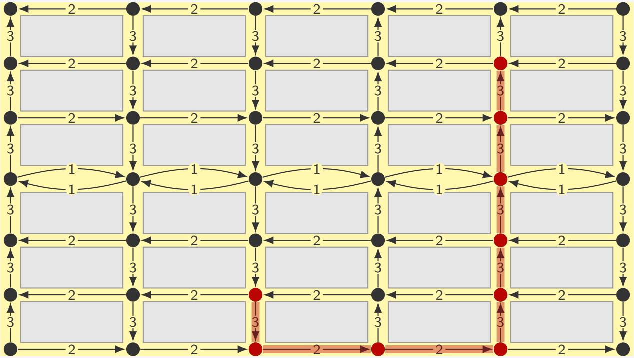


Modelamos como um digrafo com pesos nos arcos:

- Um vértice em cada cruzamento
- Um arco entre vértices consecutivos
- O peso do arco (u, v) é o tempo de viagem de u para v

Encontrando o menor caminho

Como encontrar o menor tempo para ir de A para B?



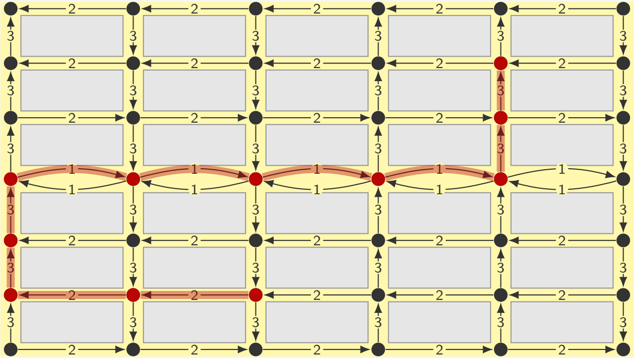
Modelamos como um digrafo com pesos nos arcos:

- Um vértice em cada cruzamento
- Um arco entre vértices consecutivos
- O peso do arco (u, v) é o tempo de viagem de u para v

Tempo de percurso do caminho: 22

Encontrando o menor caminho

Como encontrar o menor tempo para ir de A para B?



Modelamos como um digrafo com pesos nos arcos:

- Um vértice em cada cruzamento
- Um arco entre vértices consecutivos
- O peso do arco (u, v) é o tempo de viagem de u para v

Tempo de percurso do caminho: 20

Digrafos com pesos nas arestas — Representação

Como representar grafos com pesos nas arestas?

Digrafos com pesos nas arestas — Representação

Como representar grafos com pesos nas arestas?

Listas de Adjacência:

Digrafos com pesos nas arestas — Representação

Como representar grafos com pesos nas arestas?

Listas de Adjacência:

- Basta adicionar um campo `peso` no Nó da lista ligada

Digrafos com pesos nas arestas — Representação

Como representar grafos com pesos nas arestas?

Listas de Adjacência:

- Basta adicionar um campo `peso` no Nó da lista ligada

Matriz de Adjacências:

Digrafos com pesos nas arestas — Representação

Como representar grafos com pesos nas arestas?

Listas de Adjacência:

- Basta adicionar um campo `peso` no Nó da lista ligada

Matriz de Adjacências:

- Podemos indicar que não há arco usando peso 0

Digrafos com pesos nas arestas — Representação

Como representar grafos com pesos nas arestas?

Listas de Adjacência:

- Basta adicionar um campo `peso` no Nó da lista ligada

Matriz de Adjacências:

- Podemos indicar que não há arco usando peso 0
  - Isso nem sempre é uma boa opção

Digrafos com pesos nas arestas — Representação

Como representar grafos com pesos nas arestas?

Listas de Adjacência:

- Basta adicionar um campo `peso` no Nó da lista ligada

Matriz de Adjacências:

- Podemos indicar que não há arco usando peso `0`
  - Isso nem sempre é uma boa opção
  - Podemos trocar por `-1` ou então `INT_MAX`

Digrafos com pesos nas arestas — Representação

Como representar grafos com pesos nas arestas?

Listas de Adjacência:

- Basta adicionar um campo `peso` no Nó da lista ligada

Matriz de Adjacências:

- Podemos indicar que não há arco usando peso `0`
  - Isso nem sempre é uma boa opção
  - Podemos trocar por `-1` ou então `INT_MAX`
- Ou fazemos uma struct com dois campos

Digrafos com pesos nas arestas — Representação

Como representar grafos com pesos nas arestas?

Listas de Adjacência:

- Basta adicionar um campo `peso` no Nó da lista ligada

Matriz de Adjacências:

- Podemos indicar que não há arco usando peso `0`
  - Isso nem sempre é uma boa opção
  - Podemos trocar por `-1` ou então `INT_MAX`
- Ou fazemos uma struct com dois campos
  - um indica se há arco ou não

Digrafos com pesos nas arestas — Representação

Como representar grafos com pesos nas arestas?

Listas de Adjacência:

- Basta adicionar um campo `peso` no Nó da lista ligada

Matriz de Adjacências:

- Podemos indicar que não há arco usando peso `0`
  - Isso nem sempre é uma boa opção
  - Podemos trocar por `-1` ou então `INT_MAX`
- Ou fazemos uma struct com dois campos
  - um indica se há arco ou não
  - outro denota o peso do arco

Caminhos mínimos

Queremos encontrar um caminho de peso mínimo de u para v no digrafo

Caminhos mínimos

Queremos encontrar um caminho de peso mínimo de u para v no digrafo

- Consideramos que os pesos são não-negativos

Caminhos mínimos

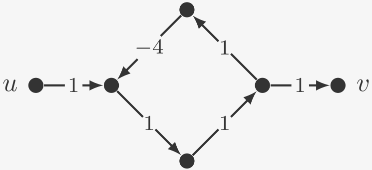
Queremos encontrar um caminho de peso mínimo de u para v no digrafo

- Consideramos que os pesos são não-negativos
- Se não, podemos querer percorrer um ciclo negativo infinitas vezes...

Caminhos mínimos

Queremos encontrar um caminho de peso mínimo de u para v no digrafo

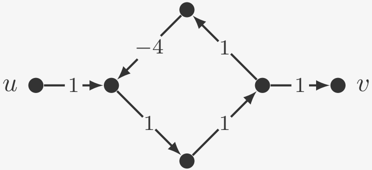
- Consideramos que os pesos são não-negativos
- Se não, podemos querer percorrer um ciclo negativo infinitas vezes...



Caminhos mínimos

Queremos encontrar um caminho de peso mínimo de u para v no digrafo

- Consideramos que os pesos são não-negativos
- Se não, podemos querer percorrer um ciclo negativo infinitas vezes...

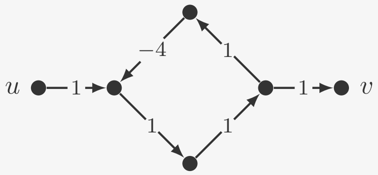


Como é o caminho mínimo de u para v?

Caminhos mínimos

Queremos encontrar um caminho de peso mínimo de u para v no digrafo

- Consideramos que os pesos são não-negativos
- Se não, podemos querer percorrer um ciclo negativo infinitas vezes...



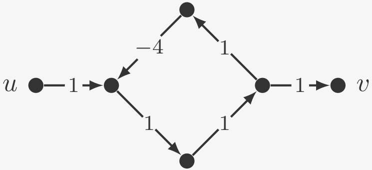
Como é o caminho mínimo de u para v?

- Ou u é vizinho de v

Caminhos mínimos

Queremos encontrar um caminho de peso mínimo de u para v no digrafo

- Consideramos que os pesos são não-negativos
- Se não, podemos querer percorrer um ciclo negativo infinitas vezes...



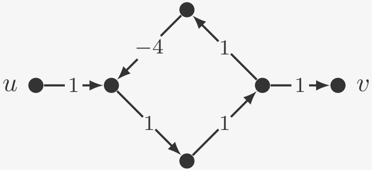
Como é o caminho mínimo de u para v?

- Ou u é vizinho de v
- Ou o caminho passa por um vizinho w de v

Caminhos mínimos

Queremos encontrar um caminho de peso mínimo de u para v no digrafo

- Consideramos que os pesos são não-negativos
- Se não, podemos querer percorrer um ciclo negativo infinitas vezes...



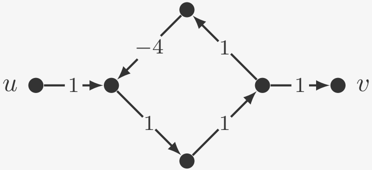
Como é o caminho mínimo de u para v?

- Ou u é vizinho de v
- Ou o caminho passa por um vizinho w de v
  - Soma do peso do caminho de u para w e de (w, v) é mínima

Caminhos mínimos

Queremos encontrar um caminho de peso mínimo de u para v no digrafo

- Consideramos que os pesos são não-negativos
- Se não, podemos querer percorrer um ciclo negativo infinitas vezes...



Como é o caminho mínimo de u para v?

- Ou u é vizinho de v
- Ou o caminho passa por um vizinho w de v
  - Soma do peso do caminho de u para w e de (w, v) é mínima
  - Este caminho de u a w tem que ter peso mínimo

Árvore de Caminhos mínimos

Árvore de caminhos mínimos (a partir de u):

Árvore de Caminhos mínimos

Árvore de caminhos mínimos (a partir de u):

- Dado u, o algoritmo encontra uma árvore enraizada em u

Árvore de Caminhos mínimos

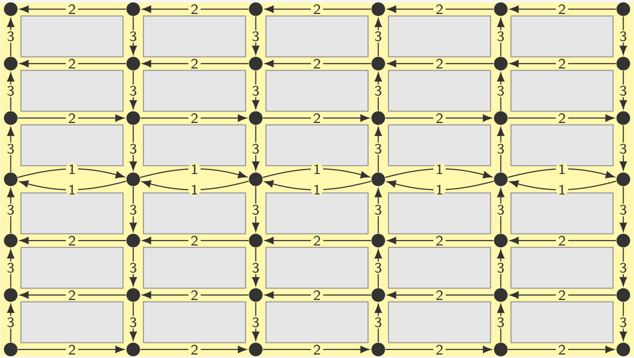
Árvore de caminhos mínimos (a partir de u):

- Dado u, o algoritmo encontra uma árvore enraizada em u
- De forma que o caminho de v para u na árvore seja um caminho mínimo de u para v no digrafo

Árvore de Caminhos mínimos

Árvore de caminhos mínimos (a partir de u):

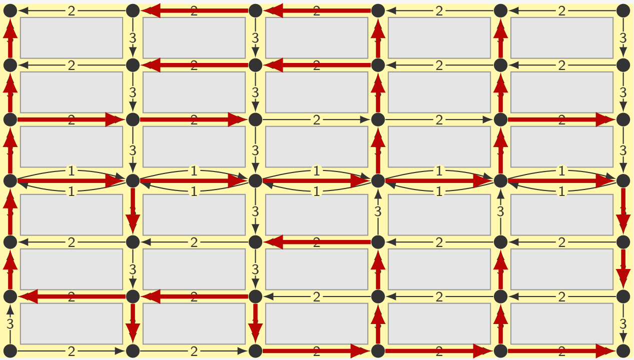
- Dado u, o algoritmo encontra uma árvore enraizada em u
- De forma que o caminho de v para u na árvore seja um caminho mínimo de u para v no digrafo



Árvore de Caminhos mínimos

Árvore de caminhos mínimos (a partir de u):

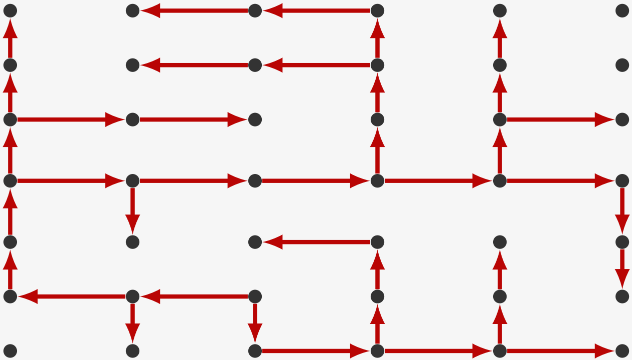
- Dado u, o algoritmo encontra uma árvore enraizada em u
- De forma que o caminho de v para u na árvore seja um caminho mínimo de u para v no digrafo



Árvore de Caminhos mínimos

Árvore de caminhos mínimos (a partir de u):

- Dado u, o algoritmo encontra uma árvore enraizada em u
- De forma que o caminho de v para u na árvore seja um caminho mínimo de u para v no digrafo



Algoritmo de Dijkstra

Em um certo momento já construímos parte da árvore

Algoritmo de Dijkstra

Em um certo momento já construímos parte da árvore

- Temos um conjunto de vértices que ainda não entraram

Algoritmo de Dijkstra

Em um certo momento já construímos parte da árvore

- Temos um conjunto de vértices que ainda não entraram
- Alguns destes são vizinhos de vértices já na árvore

Algoritmo de Dijkstra

Em um certo momento já construímos parte da árvore

- Temos um conjunto de vértices que ainda não entraram
- Alguns destes são vizinhos de vértices já na árvore
- Eles estão na franja

Algoritmo de Dijkstra

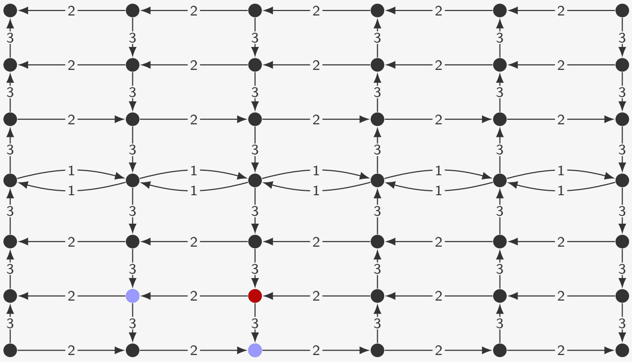
Em um certo momento já construímos parte da árvore

- Temos um conjunto de vértices que ainda não entraram
- Alguns destes são vizinhos de vértices já na árvore
- Eles estão na franja
- Pegamos o vértice na franja mais próximo de u

Algoritmo de Dijkstra

Em um certo momento já construímos parte da árvore

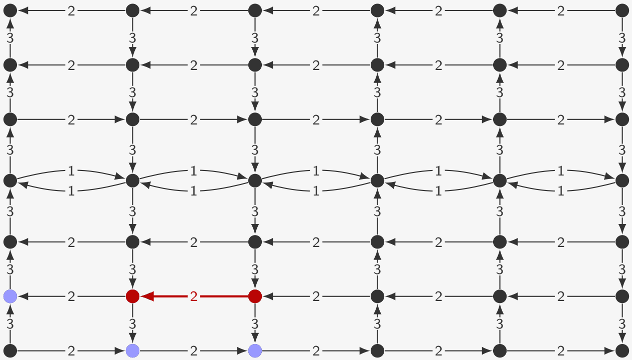
- Temos um conjunto de vértices que ainda não entraram
- Alguns destes são vizinhos de vértices já na árvore
- Eles estão na franja
- Pegamos o vértice na franja mais próximo de u



Algoritmo de Dijkstra

Em um certo momento já construímos parte da árvore

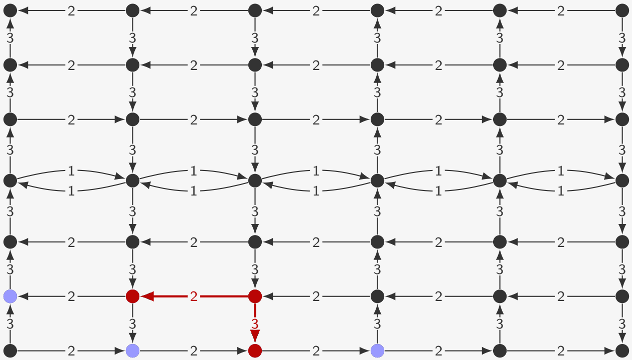
- Temos um conjunto de vértices que ainda não entraram
- Alguns destes são vizinhos de vértices já na árvore
- Eles estão na franja
- Pegamos o vértice na franja mais próximo de u



Algoritmo de Dijkstra

Em um certo momento já construímos parte da árvore

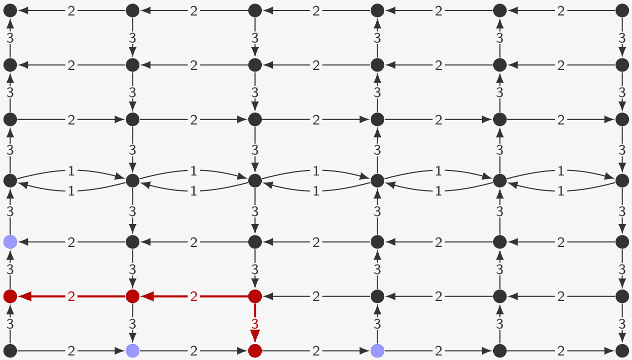
- Temos um conjunto de vértices que ainda não entraram
- Alguns destes são vizinhos de vértices já na árvore
- Eles estão na franja
- Pegamos o vértice na franja mais próximo de u



Algoritmo de Dijkstra

Em um certo momento já construímos parte da árvore

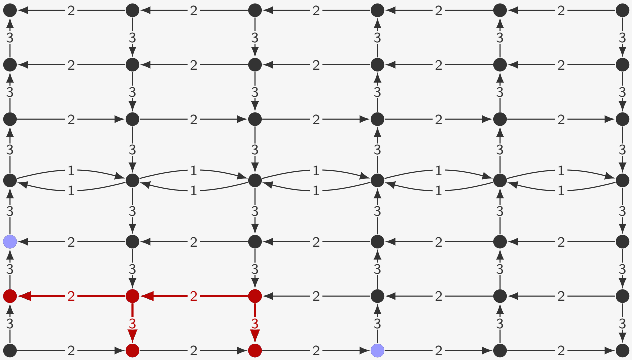
- Temos um conjunto de vértices que ainda não entraram
- Alguns destes são vizinhos de vértices já na árvore
- Eles estão na franja
- Pegamos o vértice na franja mais próximo de u



Algoritmo de Dijkstra

Em um certo momento já construímos parte da árvore

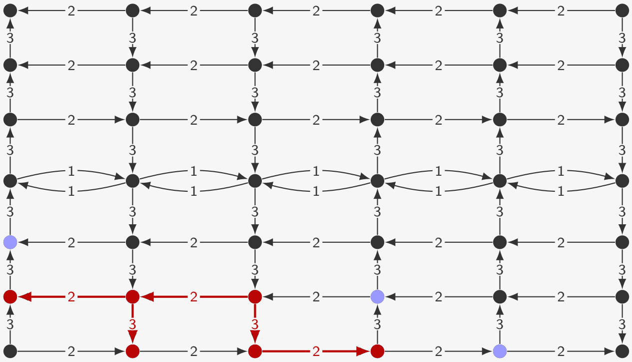
- Temos um conjunto de vértices que ainda não entraram
- Alguns destes são vizinhos de vértices já na árvore
- Eles estão na franja
- Pegamos o vértice na franja mais próximo de u



Algoritmo de Dijkstra

Em um certo momento já construímos parte da árvore

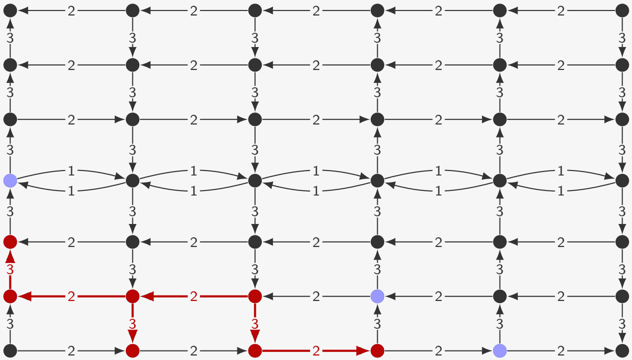
- Temos um conjunto de vértices que ainda não entraram
- Alguns destes são vizinhos de vértices já na árvore
- Eles estão na franja
- Pegamos o vértice na franja mais próximo de u



Algoritmo de Dijkstra

Em um certo momento já construímos parte da árvore

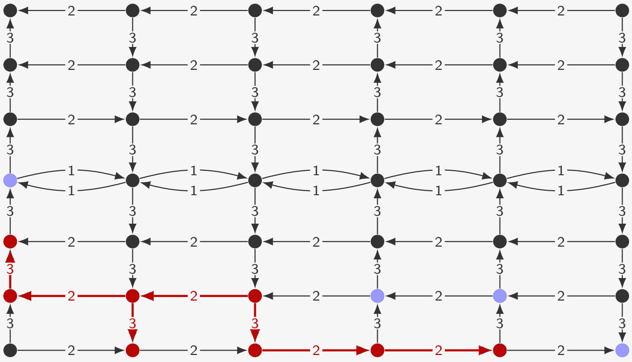
- Temos um conjunto de vértices que ainda não entraram
- Alguns destes são vizinhos de vértices já na árvore
- Eles estão na franja
- Pegamos o vértice na franja mais próximo de u



Algoritmo de Dijkstra

Em um certo momento já construímos parte da árvore

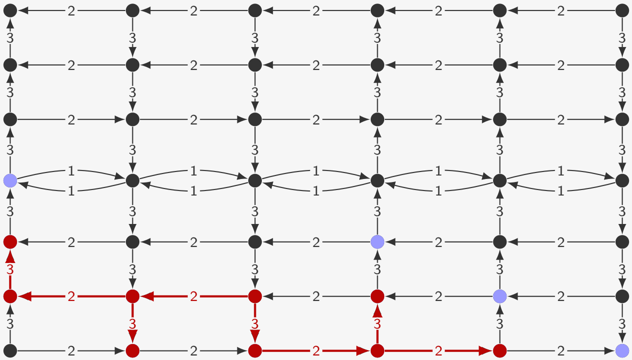
- Temos um conjunto de vértices que ainda não entraram
- Alguns destes são vizinhos de vértices já na árvore
- Eles estão na franja
- Pegamos o vértice na franja mais próximo de u



Algoritmo de Dijkstra

Em um certo momento já construímos parte da árvore

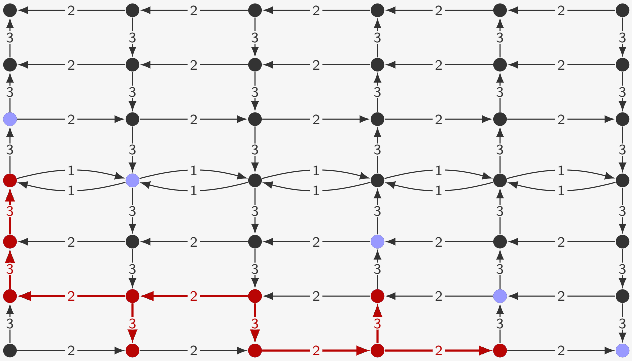
- Temos um conjunto de vértices que ainda não entraram
- Alguns destes são vizinhos de vértices já na árvore
- Eles estão na franja
- Pegamos o vértice na franja mais próximo de u



Algoritmo de Dijkstra

Em um certo momento já construímos parte da árvore

- Temos um conjunto de vértices que ainda não entraram
- Alguns destes são vizinhos de vértices já na árvore
- Eles estão na franja
- Pegamos o vértice na franja mais próximo de u



Implementação

Implementação

Grafo

```c
typedef struct no *p_no;

struct no {
  int v;
  int peso;
  p_no prox;
};

typedef struct grafo *p_grafo;

struct grafo {
  int n;
  p_no *adj;
};
```

Implementação

Grafo

```c
typedef struct no *p_no;

struct no {
  int v;
  int peso;
  p_no prox;
};

typedef struct grafo *p_grafo;

struct grafo {
  int n;
  p_no *adj;
};
```

Heap binário

```c
typedef struct {
  int prioridade;
  int vertice;
} Item;

typedef struct {
  Item *v;
  int *indice;
  int n, tamanho;
} FP;

typedef FP * p_fp;
```

Implementação

```
int * dijkstra(p_grafo g, int s) {
```

Implementação

```c
int * dijkstra(p_grafo g, int s) {
  int v, *pai = malloc(g->n * sizeof(int));
```

Implementação

```
int * dijkstra(p_grafo g, int s) {
  int v, *pai = malloc(g->n * sizeof(int));
  p_no t;
  p_fp h = criar_fprio(g->n);
```

Implementação

```
int * dijkstra(p_grafo g, int s) {
  int v, *pai = malloc(g->n * sizeof(int));
  p_no t;
  p_fp h = criar_fprio(g->n);
  for (v = 0; v < g->n; v++) {
    pai[v] = -1;
    insere(h, v, INT_MAX);
  }
```

Implementação

```
int * dijkstra(p_grafo g, int s) {
  int v, *pai = malloc(g->n * sizeof(int));
  p_no t;
  p_fp h = criar_fprio(g->n);
  for (v = 0; v < g->n; v++) {
    pai[v] = -1;
    insere(h, v, INT_MAX);
  }
  pai[s] = s;
```

Implementação

```c
int * dijkstra(p_grafo g, int s) {
  int v, *pai = malloc(g->n * sizeof(int));
  p_no t;
  p_fp h = criar_fprio(g->n);
  for (v = 0; v < g->n; v++) {
    pai[v] = -1;
    insere(h, v, INT_MAX);
  }
  pai[s] = s;
  diminuiprioridade(h, s, 0);
```

Implementação

```
int * dijkstra(p_grafo g, int s) {
  int v, *pai = malloc(g->n * sizeof(int));
  p_no t;
  p_fp h = criar_fprio(g->n);
  for (v = 0; v < g->n; v++) {
    pai[v] = -1;
    insere(h, v, INT_MAX);
  }
  pai[s] = s;
  diminuiprioridade(h, s, 0);
  while (!vazia(h)) {
```

Implementação

```c
int * dijkstra(p_grafo g, int s) {
  int v, *pai = malloc(g->n * sizeof(int));
  p_no t;
  p_fp h = criar_fprio(g->n);
  for (v = 0; v < g->n; v++) {
    pai[v] = -1;
    insere(h, v, INT_MAX);
  }
  pai[s] = s;
  diminuiprioridade(h, s, 0);
  while (!vazia(h)) {
    v = extrai_minimo(h);
```

Implementação

```
int * dijkstra(p_grafo g, int s) {
  int v, *pai = malloc(g->n * sizeof(int));
  p_no t;
  p_fp h = criar_fprio(g->n);
  for (v = 0; v < g->n; v++) {
    pai[v] = -1;
    insere(h, v, INT_MAX);
  }
  pai[s] = s;
  diminuiprioridade(h, s, 0);
  while (!vazia(h)) {
    v = extrai_minimo(h);
    if (prioridade(h, v) != INT_MAX)
```

Implementação

```
int * dijkstra(p_grafo g, int s) {
  int v, *pai = malloc(g->n * sizeof(int));
  p_no t;
  p_fp h = criar_fprio(g->n);
  for (v = 0; v < g->n; v++) {
    pai[v] = -1;
    insere(h, v, INT_MAX);
  }
  pai[s] = s;
  diminuiprioridade(h, s, 0);
  while (!vazia(h)) {
    v = extrai_minimo(h);
    if (prioridade(h, v) != INT_MAX)
      for (t = g->adj[v]; t != NULL; t = t->prox)
```

Implementação

```c
int * dijkstra(p_grafo g, int s) {
  int v, *pai = malloc(g->n * sizeof(int));
  p_no t;
  p_fp h = criar_fprio(g->n);
  for (v = 0; v < g->n; v++) {
    pai[v] = -1;
    insere(h, v, INT_MAX);
  }
  pai[s] = s;
  diminuiprioridade(h, s, 0);
  while (!vazia(h)) {
    v = extrai_minimo(h);
    if (prioridade(h, v) != INT_MAX)
      for (t = g->adj[v]; t != NULL; t = t->prox)
        if (prioridade(h, v)+t->peso < prioridade(h, t->v)) {
```

Implementação

```
int * dijkstra(p_grafo g, int s) {
  int v, *pai = malloc(g->n * sizeof(int));
  p_no t;
  p_fp h = criar_fprio(g->n);
  for (v = 0; v < g->n; v++) {
    pai[v] = -1;
    insere(h, v, INT_MAX);
  }
  pai[s] = s;
  diminuiprioridade(h, s, 0);
  while (!vazia(h)) {
    v = extrai_minimo(h);
    if (prioridade(h, v) != INT_MAX)
      for (t = g->adj[v]; t != NULL; t = t->prox)
        if (prioridade(h, v)+t->peso < prioridade(h, t->v)) {
          diminuiprioridade(h,t->v,prioridade(h,v)+t->peso);
```

Implementação

```
int * dijkstra(p_grafo g, int s) {
  int v, *pai = malloc(g->n * sizeof(int));
  p_no t;
  p_fp h = criar_fprio(g->n);
  for (v = 0; v < g->n; v++) {
    pai[v] = -1;
    insere(h, v, INT_MAX);
  }
  pai[s] = s;
  diminuiprioridade(h, s, 0);
  while (!vazia(h)) {
    v = extrai_minimo(h);
    if (prioridade(h, v) != INT_MAX)
      for (t = g->adj[v]; t != NULL; t = t->prox)
        if (prioridade(h, v)+t->peso < prioridade(h, t->v)) {
          diminuiprioridade(h,t->v,prioridade(h,v)+t->peso);
          pai[t->v] = v;
```

Implementação

```c
int * dijkstra(p_grafo g, int s) {
  int v, *pai = malloc(g->n * sizeof(int));
  p_no t;
  p_fp h = criar_fprio(g->n);
  for (v = 0; v < g->n; v++) {
    pai[v] = -1;
    insere(h, v, INT_MAX);
  }
  pai[s] = s;
  diminuiprioridade(h, s, 0);
  while (!vazia(h)) {
    v = extrai_minimo(h);
    if (prioridade(h, v) != INT_MAX)
      for (t = g->adj[v]; t != NULL; t = t->prox)
        if (prioridade(h, v)+t->peso < prioridade(h, t->v)) {
          diminuiprioridade(h,t->v,prioridade(h,v)+t->peso);
          pai[t->v] = v;
        }
  }
  return pai;
}
```

Implementação

```
int * dijkstra(p_grafo g, int s) {
  int v, *pai = malloc(g->n * sizeof(int));
  p_no t;
  p_fp h = criar_fprio(g->n);
  for (v = 0; v < g->n; v++) {
    pai[v] = -1;
    insere(h, v, INT_MAX);
  }
  pai[s] = s;
  diminuiprioridade(h, s, 0);
  while (!vazia(h)) {
    v = extrai_minimo(h);
    if (prioridade(h, v) != INT_MAX)
      for (t = g->adj[v]; t != NULL; t = t->prox)
        if (prioridade(h, v)+t->peso < prioridade(h, t->v)) {
          diminuiprioridade(h,t->v,prioridade(h,v)+t->peso);
          pai[t->v] = v;
        }
  }
  return pai;
}
```

Tempo: $O(|E| \lg |V|)$

Dúvidas?