Algorytmy i struktury danych - Algorytmy zachłanne

Marcin Żurowski

29 maja 2025

Plan zajęć

Algorytmy zachłanne

Algorytmy zachłanne:

- wykorzystywane są do rozwiązywania problemów (najczęściej optymalizacyjnych), w których należy podjąć ciąg decyzji
- nie przeglądają przestrzeni wielu możliwych rozwiązań problemu
- w każdym kroku podejmują decyzję, która w danej chwili wydaje się najkorzystniejsza
- pozwalają na znalezienie najlepszego rozwiązania tylko dla niektórych problemów

Algorytmy zachłanne

Algorytmy zachłanne:

- wykorzystywane są do rozwiązywania problemów (najczęściej optymalizacyjnych), w których należy podjąć ciąg decyzji
- nie przeglądają przestrzeni wielu możliwych rozwiązań problemu
- w każdym kroku podejmują decyzję, która w danej chwili wydaje się najkorzystniejsza
- pozwalają na znalezienie najlepszego rozwiązania tylko dla niektórych problemów

Algorytmy zachłanne

Algorytmy zachłanne:

- wykorzystywane są do rozwiązywania problemów (najczęściej optymalizacyjnych), w których należy podjąć ciąg decyzji
- nie przeglądają przestrzeni wielu możliwych rozwiązań problemu
- w każdym kroku podejmują decyzję, która w danej chwili wydaje się najkorzystniejsza
- pozwalają na znalezienie najlepszego rozwiązania tylko dla niektórych problemów

Algorytmy zachłanne

Algorytmy zachłanne:

- wykorzystywane są do rozwiązywania problemów (najczęściej optymalizacyjnych), w których należy podjąć ciąg decyzji
- nie przeglądają przestrzeni wielu możliwych rozwiązań problemu
- w każdym kroku podejmują decyzję, która w danej chwili wydaje się najkorzystniejsza
- pozwalają na znalezienie najlepszego rozwiązania tylko dla niektórych problemów

Algorytmy zachłanne

Algorytmy zachłanne:

- wykorzystywane są do rozwiązywania problemów (najczęściej optymalizacyjnych), w których należy podjąć ciąg decyzji
- nie przeglądają przestrzeni wielu możliwych rozwiązań problemu
- w każdym kroku podejmują decyzję, która w danej chwili wydaje się najkorzystniejsza
- pozwalają na znalezienie najlepszego rozwiązania tylko dla niektórych problemów

Algorytmy zachłanne

Przykład 1: problem wyboru zajęć

- Dany jest zbiór $S = 1, 2, \ldots, n$ składający się z n proponowanych zajęć. Dysponujemy jedną salą wykładową, w której mogą odbywać się w danej chwili tylko jedne zajęcia. Każde zajęcia mają określony czas rozpoczęcia s_i oraz czas zakończenia f_i. Zajęcia o numerze i odbywają się w przedziale czasu $[s_i, f_i)$. Należy wybrać jak największy zbiór zajęć, które ze sobą nie kolidują.
- Strategia zachłanna: zakładamy, że zajęcia są posortowane ze względu na czas zakończenia, tak że $f_1 \leq f_2 \leq \ldots \leq f_n$. Przeglądamy je kolejno. W każdym kroku algorytmu sprawdzamy, czy bieżące zajęcia kolidują z wybranymi wcześniej. Jeśli nie, to dołączamy je do zbioru wybranych zajęć.

Algorytmy zachłanne

Przykład 1: problem wyboru zajęć

- Dany jest zbiór $S = 1, 2, \ldots, n$ składający się z n proponowanych zajęć. Dysponujemy jedną salą wykładową, w której mogą odbywać się w danej chwili tylko jedne zajęcia. Każde zajęcia mają określony czas rozpoczęcia s_i oraz czas zakończenia f_i. Zajęcia o numerze i odbywają się w przedziale czasu $[s_i, f_i)$. Należy wybrać jak największy zbiór zajęć, które ze sobą nie kolidują.
- Strategia zachłanna: zakładamy, że zajęcia są posortowane ze względu na czas zakończenia, tak że $f_1 \leq f_2 \leq \cdots \leq f_n$. Przeglądamy je kolejno. W każdym kroku algorytmu sprawdzamy, czy bieżące zajęcia kolidują z wybranymi wcześniej. Jeśli nie, to dołączamy je do zbioru wybranych zajęć.

Algorytmy zachłanne

Przykład 1: problem wyboru zajęć

- Dany jest zbiór $S = 1, 2, \ldots, n$ składający się z n proponowanych zajęć. Dysponujemy jedną salą wykładową, w której mogą odbywać się w danej chwili tylko jedne zajęcia. Każde zajęcia mają określony czas rozpoczęcia s_i oraz czas zakończenia f_i. Zajęcia o numerze i odbywają się w przedziale czasu $[s_i, f_i)$. Należy wybrać jak największy zbiór zajęć, które ze sobą nie kolidują.
- Strategia zachłanna: zakładamy, że zajęcia są posortowane ze względu na czas zakończenia, tak że $f_1 \leq f_2 \leq \cdots \leq f_n$. Przeglądamy je kolejno. W każdym kroku algorytmu sprawdzamy, czy bieżące zajęcia kolidują z wybranymi wcześniej. Jeśli nie, to dołączamy je do zbioru wybranych zajęć.

Algorytmy zachłanne

```
procedure GREEDY-ACTIVITY-SELECTOR(s,f,n)
 A = {1}
 j = 1
 for i = 2 to n
  if s_i ≥ f_j
    A = A ∪ {i}
    j = i
 return A
```

Twierdzenie

Algorytm GREEDY-ACTIVITY-SELECTOR generuje rozwiązanie problemu wyboru zajęć o największym rozmiarze.

Algorytmy zachłanne

```
procedure GREEDY-ACTIVITY-SELECTOR(s,f,n)
 A = {1}
 j = 1
 for i = 2 to n
  if s_i ≥ f_j
   A = A ∪ {i}
   j = i
 return A
```

Twierdzenie

Algorytm GREEDY-ACTIVITY-SELECTOR generuje rozwiązanie problemu wyboru zajęć o największym rozmiarze.

Algorytmy zachłanne

Przykład 2: problem plecakowy

Mamy plecak, w którym można przenieść przedmioty o łącznej wadze nieprzekraczającej M. Danych jest n przedmiotów $e_1, \ldots, e_n$, przy czym przedmiot e_i ma wartość p_i i wagę m_i. Znaleźć jak najcenniejszy zbiór przedmiotów, który można przenieść w plecaku.

- Wersja ciągła: można brać ułamkowe części przedmiotów. Poprawny jest algorytm zachłanny polegający na posortowaniu przedmiotów nierosnąco według wartości $\frac{p_i}{m_i}$ i wkładaniu ich kolejno do plecaka aż do zapełnienia go (w tym celu można wziąć ułamkową część ostatniego wybranego przedmiotu)
- Wersja zwykła (dyskretna): przedmiotów nie można dzielić na części. Powyższa strategia zachłanna nie jest poprawna.

Algorytmy zachłanne

Przykład 2: problem plecakowy
Mamy plecak, w którym można przenieść przedmioty o łącznej wadze nieprzekraczającej M. Danych jest n przedmiotów $e_1, \ldots, e_n$, przy czym przedmiot e_i ma wartość p_i i wagę m_i. Znaleźć jak najcenniejszy zbiór przedmiotów, który można przenieść w plecaku.

- Wersja ciągła: można brać ułamkowe części przedmiotów. Poprawny jest algorytm zachłanny polegający na posortowaniu przedmiotów nierosnąco według wartości $\frac{p_i}{m_i}$ i wkładaniu ich kolejno do plecaka aż do zapełnienia go (w tym celu można wziąć ułamkową część ostatniego wybranego przedmiotu)
- Wersja zwykła (dyskretna): przedmiotów nie wolno dzielić na części. Powyższa strategia zachłanna nie jest poprawna.

Algorytmy zachłanne

Przykład 2: problem plecakowy
Mamy plecak, w którym można przenieść przedmioty o łącznej wadze nieprzekraczającej M. Danych jest n przedmiotów $e_1, \ldots, e_n$, przy czym przedmiot e_i ma wartość p_i i wagę m_i. Znaleźć jak najcenniejszy zbiór przedmiotów, który można przenieść w plecaku.

- Wersja ciągła: można brać ułamkowe części przedmiotów. Poprawny jest algorytm zachłanny polegający na posortowaniu przedmiotów nierosnąco według wartości $\frac{p_i}{m_i}$ i wkładaniu ich kolejno do plecaka aż do zapełnienia go (w tym celu można wziąć ułamkową część ostatniego wybranego przedmiotu)
- Wersja zwykła (dyskretna): przedmiotów nie wolno dzielić na części. Powyższa strategia zachłanna nie jest poprawna.

Algorytmy zachłanne

Przykład 2: problem plecakowy
Mamy plecak, w którym można przenieść przedmioty o łącznej wadze nieprzekraczającej M. Danych jest n przedmiotów $e_1, \ldots, e_n$, przy czym przedmiot e_i ma wartość p_i i wagę m_i. Znaleźć jak najcenniejszy zbiór przedmiotów, który można przenieść w plecaku.

- Wersja ciągła: można brać ułamkowe części przedmiotów. Poprawny jest algorytm zachłanny polegający na posortowaniu przedmiotów nierosnąco według wartości $\frac{p_i}{m_i}$ i wkładaniu ich kolejno do plecaka aż do zapełnienia go (w tym celu można wziąć ułamkową część ostatniego wybranego przedmiotu)
- Wersja zwykła (dyskretna): przedmiotów nie wolno dzielić na części. Powyższa strategia zachłanna nie jest poprawna.

Algorytmy zachłanne

Przykład 3: problem podziału zbioru

Dany jest zbiór A składający się z n liczb naturalnych $a_1, \ldots, a_n$. Należy podzielić go na dwa rozłączne podzbiory A_1 i A_2 ($A_1 \cup A_2 = A$) w taki sposób, żeby wartość bezwzględna różnicy między sumą elementów zbioru A_1 a sumą elementów zbioru A_2 była jak najmniejsza

Niepoprawne strategie zachłanne:

- Posortuj liczby nierosnąco, przydzielaj kolejno do zbiorów A_1 i A_2, tak żeby różnica między sumami ich elementów była jak najmniejsza. Kontrprzykład: $A = \{3, 3, 2, 2, 2\}$
- Posortuj liczby nierosnąco, każdą kolejną liczbę dołóż do zbioru A_1, o ile nie spowoduje to, że suma jego elementów przekroczy $\frac{\sum_{i=1}^{n} a_i}{2}$. W przeciwnym razie dołóż ją do zbioru A_2. Kontrprzykład: $A = \{5, 3, 3, 2, 2\}$

Algorytmy zachłanne

Przykład 3: problem podziału zbioru
Dany jest zbiór A składający się z n liczb naturalnych $a_1, \ldots, a_n$. Należy podzielić go na dwa rozłączne podzbiory A_1 i A_2 ($A_1 \cup A_2 = A$) w taki sposób, żeby wartość bezwzględna różnicy między sumą elementów zbioru A_1 a sumą elementów zbioru A_2 była jak najmniejsza

Niepoprawne strategie zachłanne:

- Posortuj liczby nierosnąco, przydzielaj kolejno do zbiorów A_1 i A_2, tak żeby różnica między sumami ich elementów była jak najmniejsza. Kontrprzykład: $A = \{3, 3, 2, 2, 2\}$
- Posortuj liczby nierosnąco. Każdą kolejną liczbę dołącz do zbioru A_1, o ile nie spowoduje to, że suma jego elementów przekroczy $\frac{\sum_{i=1}^{n} a_i}{2}$. W przeciwnym razie dołącz tę liczbę do zbioru A_2. Kontrprzykład: $A = \{5, 3, 3, 3, 2, 2\}$

Algorytmy zachłanne

Przykład 3: problem podziału zbioru
Dany jest zbiór A składający się z n liczb naturalnych $a_1, \ldots, a_n$. Należy podzielić go na dwa rozłączne podzbiory A_1 i A_2 ($A_1 \cup A_2 = A$) w taki sposób, żeby wartość bezwzględna różnicy między sumą elementów zbioru A_1 a sumą elementów zbioru A_2 była jak najmniejsza
Niepoprawne strategie zachłanne:

- Posortuj liczby nierosnąco, przydzielaj kolejno do zbiorów A_1 i A_2, tak żeby różnica między sumami ich elementów była jak najmniejsza. Kontrprzykład: $A = \{3, 3, 2, 2, 2\}$
- Posortuj liczby nierosnąco. Każdą kolejną liczbę dołącz do zbioru A_1, o ile nie spowoduje to, że suma jego elementów przekroczy $\frac{\sum_{i=1}^{n} a_i}{2}$. W przeciwnym razie dołącz tę liczbę do zbioru A_2. Kontrprzykład: $A = \{5, 3, 3, 3, 2, 2\}$

Algorytmy zachłanne

Przykład 3: problem podziału zbioru
Dany jest zbiór A składający się z n liczb naturalnych $a_1, \ldots, a_n$. Należy podzielić go na dwa rozłączne podzbiory A_1 i A_2 ($A_1 \cup A_2 = A$) w taki sposób, żeby wartość bezwzględna różnicy między sumą elementów zbioru A_1 a sumą elementów zbioru A_2 była jak najmniejsza
Niepoprawne strategie zachłanne:

- Posortuj liczby nierosnąco, przydzielaj kolejno do zbiorów A_1 i A_2, tak żeby różnica między sumami ich elementów była jak najmniejsza. Kontrprzykład: $A = \{3, 3, 2, 2, 2\}$
- Posortuj liczby nierosnąco. Każdą kolejną liczbę dołącz do zbioru A_1, o ile nie spowoduje to, że suma jego elementów przekroczy $\frac{\sum_{i=1}^{n} a_i}{2}$. W przeciwnym razie dołącz tę liczbę do zbioru A_2. Kontrprzykład: $A = \{5, 3, 3, 3, 2, 2\}$

Algorytmy zachłanne

Problemy, dla których istnieje poprawny algorytm zachłanny, posiadają własność wyboru zachłannego oraz optymalną podstrukturę

- Własność wyboru zachłannego oznacza, że za pomocą lokalnie optymalnych (zachłannych) wyborów można uzyskać optymalne rozwiązanie
- Problem wykazuje optymalną podstrukturę, jeśli rozwiązanie optymalne jest funkcją optymalnych rozwiązań podproblemów

Algorytmy zachłanne

Problemy, dla których istnieje poprawny algorytm zachłanny, posiadają własność wyboru zachłannego oraz optymalną podstrukturę

- Własność wyboru zachłannego oznacza, że za pomocą lokalnie optymalnych (zachłannych) wyborów można uzyskać optymalne rozwiązanie
- Problem wykazuje optymalną podstrukturę, jeśli rozwiązanie optymalne jest funkcją optymalnych rozwiązań podproblemów

Algorytmy zachłanne

Problemy, dla których istnieje poprawny algorytm zachłanny, posiadają własność wyboru zachłannego oraz optymalną podstrukturę

- Własność wyboru zachłannego oznacza, że za pomocą lokalnie optymalnych (zachłannych) wyborów można uzyskać optymalne rozwiązanie
- Problem wykazuje optymalną podstrukturę, jeśli rozwiązanie optymalne jest funkcją optymalnych rozwiązań podproblemów

Algorytmy zachłanne

Minimalne drzewo rozpinające (minimum spanning tree, MST)

- Dany jest graf $G = (V, E)$. Z każdą krawędzią $(u, v) \in E$ związana jest waga $w(u, v)$. Należy znaleźć drzewo $T = (V, F)$, gdzie $F \subseteq E$ (drzewo rozpinające grafu G), dla którego łączna waga

$$w(F) = \sum_{(u,v) \in F} w(u, v)$$

jest najmniejsza

Problem znajdowania minimalnego drzewa rozpinającego występuje np. przy projektowaniu układów elektronicznych.

Algorytmy zachłanne

Minimalne drzewo rozpinające (minimum spanning tree, MST)

- Dany jest graf $G = (V, E)$. Z każdą krawędzią $(u, v) \in E$ związana jest waga $w(u, v)$. Należy znaleźć drzewo $T = (V, F)$, gdzie $F \subseteq E$ (drzewo rozpinające grafu G), dla którego łączna waga

$$w(F) = \sum_{(u,v) \in F} w(u, v)$$

  jest najmniejsza

Problem znajdowania minimalnego drzewa rozpinającego występuje np. przy projektowaniu układów elektronicznych.

Algorytmy zachłanne

Minimalne drzewo rozpinające (minimum spanning tree, MST)

- Dany jest graf $G = (V, E)$. Z każdą krawędzią $(u, v) \in E$ związana jest waga $w(u, v)$. Należy znaleźć drzewo $T = (V, F)$, gdzie $F \subseteq E$ (drzewo rozpinające grafu G), dla którego łączna waga

$$w(F) = \sum_{(u,v) \in F} w(u, v)$$

  jest najmniejsza

Problem znajdowania minimalnego drzewa rozpinającego występuje np. przy projektowaniu układów elektronicznych.

Algorytmy zachłanne

Minimalne drzewo rozpinające (minimum spanning tree, MST)

- Dany jest graf $G = (V, E)$. Z każdą krawędzią $(u, v) \in E$ związana jest waga $w(u, v)$. Należy znaleźć drzewo $T = (V, F)$, gdzie $F \subseteq E$ (drzewo rozpinające grafu G), dla którego łączna waga

$$w(F) = \sum_{(u,v) \in F} w(u, v)$$

  jest najmniejsza

Problem znajdowania minimalnego drzewa rozpinającego występuje np. przy projektowaniu układów elektronicznych.

Algorytmy zachłanne

Algorytm generyczny - strategia zachłanna

```
procedure GENERIC-MST(G,w)
 F = φ
 while (V, F) nie tworzy drzewa rozpinającego
  znajdź krawędź (u, v), która jest bezpieczna dla F
  F = F ∪ {(u, v)}
 return F
```

Krawędź bezpieczna dla F to krawędź, która nie powoduje naruszenia warunku, że zbiór F jest podzbiorem zbioru krawędzi pewnego minimalnego drzewa rozpinającego w G

Algorytmy zachłanne

Algorytm generyczny - strategia zachłanna

```
procedure GENERIC-MST(G,w)
 F = φ
 while (V, F) nie tworzy drzewa rozpinającego
  znajdź krawędź (u, v), która jest bezpieczna dla F
  F = F ∪ {(u, v)}
 return F
```

Krawędź bezpieczna dla F to krawędź, która nie powoduje naruszenia warunku, że zbiór F jest podzbiorem zbioru krawędzi pewnego minimalnego drzewa rozpinającego w G

Algorytmy zachłanne

Algorytm generyczny - strategia zachłanna

```
procedure GENERIC-MST(G,w)
 F = φ
 while (V, F) nie tworzy drzewa rozpinającego
  znajdź krawędź (u, v), która jest bezpieczna dla F
  F = F ∪ {(u, v)}
 return F
```

Krawędź bezpieczna dla F to krawędź, która nie powoduje naruszenia warunku, że zbiór F jest podzbiorem zbioru krawędzi pewnego minimalnego drzewa rozpinającego w G

Algorytmy zachłanne

- Krawędź o danej własności A jest krawędzią lekką, jeśli jej waga jest najmniejsza spośród wag wszystkich krawędzi o własności A
- W algorytmie GENERIC-MST krawędzią bezpieczną jest krawędź lekka o pewnej własności A. W zależności od wyboru tej własności, otrzymujemy różne algorytmy konstruowania MST
- Algorytm Kruskala: wybieramy krawędź lekką, która nie powoduje powstania cyklu w (V, F)
- Algorytm Prima: wybieramy krawędź lekką, która nie powoduje powstania cyklu w (V, F) i nie narusza spójności zbioru krawędzi F

Algorytmy zachłanne

- Krawędź o danej własności A jest krawędzią lekką, jeśli jej waga jest najmniejsza spośród wag wszystkich krawędzi o własności A
- W algorytmie GENERIC-MST krawędzią bezpieczną jest krawędź lekka o pewnej własności A. W zależności od wyboru tej własności, otrzymujemy różne algorytmy konstruowania MST
- Algorytm Kruskala: wybieramy krawędź lekką, która nie powoduje powstania cyklu w (V, F)
- Algorytm Prima: wybieramy krawędź lekką, która nie powoduje powstania cyklu w (V, F) i nie narusza spójności zbioru krawędzi F

Algorytmy zachłanne

- Krawędź o danej własności A jest krawędzią lekką, jeśli jej waga jest najmniejsza spośród wag wszystkich krawędzi o własności A
- W algorytmie GENERIC-MST krawędzią bezpieczną jest krawędź lekka o pewnej własności A. W zależności od wyboru tej własności, otrzymujemy różne algorytmy konstruowania MST
- Algorytm Kruskala: wybieramy krawędź lekką, która nie powoduje powstania cyklu w (V, F)
- Algorytm Prima: wybieramy krawędź lekką, która nie powoduje powstania cyklu w (V, F) i nie narusza spójności zbioru krawędzi F

Algorytmy zachłanne

- Krawędź o danej własności A jest krawędzią lekką, jeśli jej waga jest najmniejsza spośród wag wszystkich krawędzi o własności A
- W algorytmie GENERIC-MST krawędzią bezpieczną jest krawędź lekka o pewnej własności A. W zależności od wyboru tej własności, otrzymujemy różne algorytmy konstruowania MST
- Algorytm Kruskala: wybieramy krawędź lekką, która nie powoduje powstania cyklu w (V, F)
- Algorytm Prima: wybieramy krawędź lekką, która nie powoduje powstania cyklu w (V, F) i nie narusza spójności zbioru krawędzi F

Algorytmy zachłanne

Algorytm Kruskala:

- Strategia zachłanna: przeglądamy krawędzie w kolejności niemalejących wag. Każdą rozważaną krawędź dodajemy do zbioru F, o ile nie spowoduje to utworzenia w nim cyklu
- Aby łatwo sprawdzić, czy dodanie krawędzi tworzy cykl w zbiorze F, przechowujemy informację o składowych spójności grafu (V, F). Stosujemy strukturę danych dla zbiorów rozłącznych, umożliwiającą wykonanie następujących operacji:
  - Strukturę dla zbiorów rozłącznych można zaimplementować na różne sposoby, wykorzystując listy z dowiązaniami lub las

Algorytmy zachłanne

Algorytm Kruskala:

- Strategia zachłanna: przeglądamy krawędzie w kolejności niemalejących wag. Każdą rozważaną krawędź dodajemy do zbioru F, o ile nie spowoduje to utworzenia w nim cyklu
- Aby łatwo sprawdzić, czy dodanie krawędzi tworzy cykl w zbiorze F, przechowujemy informację o składowych spójności grafu (V, F). Stosujemy strukturę danych dla zbiorów rozłącznych, umożliwiającą wykonanie następujących operacji:
  - stworzenie jednoelementowego zbioru zawierającego wierzchołek v
  - zwrócenie zbioru zawierającego wierzchołek v
  - połączenie zbiorów zawierających wierzchołki u i v
- Strukturę dla zbiorów rozłącznych można zaimplementować na różne sposoby, wykorzystując listy z dowiązaniami lub drzewa

Algorytmy zachłanne

Algorytm Kruskala:

- Strategia zachłanna: przeglądamy krawędzie w kolejności niemalejących wag. Każdą rozważaną krawędź dodajemy do zbioru F, o ile nie spowoduje to utworzenia w nim cyklu
- Aby łatwo sprawdzić, czy dodanie krawędzi tworzy cykl w zbiorze F, przechowujemy informację o składowych spójności grafu (V, F). Stosujemy strukturę danych dla zbiorów rozłącznych, umożliwiającą wykonanie następujących operacji:
  - stworzenie jednoelementowego zbioru zawierającego wierzchołek v
  - zwrócenie zbioru zawierającego wierzchołek v
  - połączenie zbiorów zawierających wierzchołki u i v
- Strukturę dla zbiorów rozłącznych można zaimplementować na różne sposoby, wykorzystując listy z dowiązaniami lub drzewa

Algorytmy zachłanne

Algorytm Kruskala:

- Strategia zachłanna: przeglądamy krawędzie w kolejności niemalejących wag. Każdą rozważaną krawędź dodajemy do zbioru F, o ile nie spowoduje to utworzenia w nim cyklu
- Aby łatwo sprawdzić, czy dodanie krawędzi tworzy cykl w zbiorze F, przechowujemy informację o składowych spójności grafu (V, F). Stosujemy strukturę danych dla zbiorów rozłącznych, umożliwiającą wykonanie następujących operacji:
  - stworzenie jednoelementowego zbioru zawierającego wierzchołek v
  - zwrócenie zbioru zawierającego wierzchołek v
  - połączenie zbiorów zawierających wierzchołki u i v
- Strukturę dla zbiorów rozłącznych można zaimplementować na różne sposoby, wykorzystując listy z dowiązaniami lub drzewa

Algorytmy zachłanne

Algorytm Kruskala:

- Strategia zachłanna: przeglądamy krawędzie w kolejności niemalejących wag. Każdą rozważaną krawędź dodajemy do zbioru F, o ile nie spowoduje to utworzenia w nim cyklu
- Aby łatwo sprawdzić, czy dodanie krawędzi tworzy cykl w zbiorze F, przechowujemy informację o składowych spójności grafu (V, F). Stosujemy strukturę danych dla zbiorów rozłącznych, umożliwiającą wykonanie następujących operacji:
  - stworzenie jednoelementowego zbioru zawierającego wierzchołek v
  - zwrócenie zbioru zawierającego wierzchołek v
  - połączenie zbiorów zawierających wierzchołki u i v
- Strukturę dla zbiorów rozłącznych można zaimplementować na różne sposoby, wykorzystując listy z dowiązaniami lub drzewa

Algorytmy zachłanne

Algorytm Kruskala:

- Strategia zachłanna: przeglądamy krawędzie w kolejności niemalejących wag. Każdą rozważaną krawędź dodajemy do zbioru F, o ile nie spowoduje to utworzenia w nim cyklu
- Aby łatwo sprawdzić, czy dodanie krawędzi tworzy cykl w zbiorze F, przechowujemy informację o składowych spójności grafu (V, F). Stosujemy strukturę danych dla zbiorów rozłącznych, umożliwiającą wykonanie następujących operacji:
  - stworzenie jednoelementowego zbioru zawierającego wierzchołek v
  - zwrócenie zbioru zawierającego wierzchołek v
  - połączenie zbiorów zawierających wierzchołki u i v
- Strukturę dla zbiorów rozłącznych można zaimplementować na różne sposoby, wykorzystując listy z dowiązaniami lub drzewa

Algorytmy zachłanne

Algorytm Kruskala:

- Strategia zachłanna: przeglądamy krawędzie w kolejności niemalejących wag. Każdą rozważaną krawędź dodajemy do zbioru F, o ile nie spowoduje to utworzenia w nim cyklu
- Aby łatwo sprawdzić, czy dodanie krawędzi tworzy cykl w zbiorze F, przechowujemy informację o składowych spójności grafu (V, F). Stosujemy strukturę danych dla zbiorów rozłącznych, umożliwiającą wykonanie następujących operacji:
  - stworzenie jednoelementowego zbioru zawierającego wierzchołek v
  - zwrócenie zbioru zawierającego wierzchołek v
  - połączenie zbiorów zawierających wierzchołki u i v
- Strukturę dla zbiorów rozłącznych można zaimplementować na różne sposoby, wykorzystując listy z dowiązaniami lub drzewa

Algorytmy zachłanne

```
procedure MST-KRUSKAL(G,w)
 F = φ
 utwórz rozłączne podzbiory wierzchołków:
{v_1}, {v_2}, ..., {v_n},
 posortuj krawędzie z E niemalejąco względem wag w
 while nie wszystkie podzbiory zostaną scalone
  wybierz kolejną najlżejszą krawędź (u, v)
  if (u, v) łączy wierzchołki z różnych zbiorów
   F = F ∪ {(u, v)}
   scal zbiory zawierające wierzchołki u, v
 return F
```

Algorytmy zachłanne

Algorytm Prima:

- Algorytm rozpoczyna od wierzchołka startowego r
- W każdym kroku algorytmu zachłannie wybieramy najlżejszą krawędź dołączającą do drzewa nowy wierzchołek
- Aby łatwo wybierać najlżejsze krawędzie, wykorzystujemy kolejkę priorytetową Q typu min zawierającą wierzchołki grafu. Każdy wierzchołek v ma atrybut $v.key$ (klucz), określający minimalny koszt przyłączenia wierzchołka v do bieżącego drzewa. Porównywanie elementów kolejki Q polega na porównywaniu ich kluczy
- Zmniejszenie wartości $v.key$ oznacza zatem wywołanie na kolejce Q procedury DECREASE-KEY
- Każdy wierzchołek v posiada także atrybut $v.pred$, oznaczający wierzchołek, do którego przyłączany jest w danym momencie wierzchołek v
- Minimalne drzewo rozpinające tworzy [illegible]

Algorytmy zachłanne

Algorytm Prima:

- Algorytm rozpoczyna od wierzchołka startowego r
- W każdym kroku algorytmu zachłannie wybieramy najlżejszą krawędź dołączającą do drzewa nowy wierzchołek
- Aby łatwo wybierać najlżejsze krawędzie, wykorzystujemy kolejkę priorytetową Q typu min zawierającą wierzchołki grafu. Każdy wierzchołek v ma atrybut $v.key$ (klucz), określający minimalny koszt przyłączenia wierzchołka v do bieżącego drzewa. Porównywanie elementów kolejki Q polega na porównywaniu ich kluczy
- Zmniejszenie wartości $v.key$ oznacza zatem wywołanie na kolejce Q procedury DECREASE-KEY
- Każdy wierzchołek v posiada także atrybut $v.prev$, oznaczający wierzchołek, do którego przyłączany jest w drzewie wierzchołek v
- Minimalne drzewo rozpinające tworzą krawędzie $(v, v.prev)$, gdzie $v \in V \setminus \{r\}$

Algorytmy zachłanne

Algorytm Prima:

- Algorytm rozpoczyna od wierzchołka startowego r
- W każdym kroku algorytmu zachłannie wybieramy najlżejszą krawędź dołączającą do drzewa nowy wierzchołek
- Aby łatwo wybierać najlżejsze krawędzie, wykorzystujemy kolejkę priorytetową Q typu min zawierającą wierzchołki grafu. Każdy wierzchołek v ma atrybut $v.key$ (klucz), określający minimalny koszt przyłączenia wierzchołka v do bieżącego drzewa. Porównywanie elementów kolejki Q polega na porównywaniu ich kluczy
- Zmniejszenie wartości $v.key$ oznacza zatem wywołanie na kolejce Q procedury DECREASE-KEY
- Każdy wierzchołek v posiada także atrybut $v.prev$, oznaczający wierzchołek, do którego przyłączany jest w drzewie wierzchołek v
- Minimalne drzewo rozpinające tworzą krawędzie $(v, v.prev)$, gdzie $v \in V \setminus \{r\}$

Algorytmy zachłanne

Algorytm Prima:

- Algorytm rozpoczyna od wierzchołka startowego r
- W każdym kroku algorytmu zachłannie wybieramy najlżejszą krawędź dołączającą do drzewa nowy wierzchołek
- Aby łatwo wybierać najlżejsze krawędzie, wykorzystujemy kolejkę priorytetową Q typu min zawierającą wierzchołki grafu. Każdy wierzchołek v ma atrybut $v.key$ (klucz), określający minimalny koszt przyłączenia wierzchołka v do bieżącego drzewa. Porównywanie elementów kolejki Q polega na porównywaniu ich kluczy
- Zmniejszenie wartości $v.key$ oznacza zatem wywołanie na kolejce Q procedury DECREASE-KEY
- Każdy wierzchołek v posiada także atrybut $v.prev$, oznaczający wierzchołek, do którego przyłączany jest w drzewie wierzchołek v
- Minimalne drzewo rozpinające tworzą krawędzie $(v, v.prev)$, gdzie $v \in V \setminus \{r\}$

Algorytmy zachłanne

Algorytm Prima:

- Algorytm rozpoczyna od wierzchołka startowego r
- W każdym kroku algorytmu zachłannie wybieramy najlżejszą krawędź dołączającą do drzewa nowy wierzchołek
- Aby łatwo wybierać najlżejsze krawędzie, wykorzystujemy kolejkę priorytetową Q typu min zawierającą wierzchołki grafu. Każdy wierzchołek v ma atrybut $v.key$ (klucz), określający minimalny koszt przyłączenia wierzchołka v do bieżącego drzewa. Porównywanie elementów kolejki Q polega na porównywaniu ich kluczy
- Zmniejszenie wartości $v.key$ oznacza zatem wywołanie na kolejce Q procedury DECREASE-KEY
- Każdy wierzchołek v posiada także atrybut $v.prev$, oznaczający wierzchołek, do którego przyłączany jest w drzewie wierzchołek v
- Minimalne drzewo rozpinające tworzą krawędzie $(v, v.prev)$, gdzie $v \in V \setminus \{r\}$

Algorytmy zachłanne

Algorytm Prima:

- Algorytm rozpoczyna od wierzchołka startowego r
- W każdym kroku algorytmu zachłannie wybieramy najlżejszą krawędź dołączającą do drzewa nowy wierzchołek
- Aby łatwo wybierać najlżejsze krawędzie, wykorzystujemy kolejkę priorytetową Q typu min zawierającą wierzchołki grafu. Każdy wierzchołek v ma atrybut $v.key$ (klucz), określający minimalny koszt przyłączenia wierzchołka v do bieżącego drzewa. Porównywanie elementów kolejki Q polega na porównywaniu ich kluczy
- Zmniejszenie wartości $v.key$ oznacza zatem wywołanie na kolejce Q procedury DECREASE-KEY
- Każdy wierzchołek v posiada także atrybut $v.prev$, oznaczający wierzchołek, do którego przyłączany jest w drzewie wierzchołek v
- Minimalne drzewo rozpinające tworzą krawędzie $(v, v.prev)$, gdzie $v \in V \setminus \{r\}$

Algorytmy zachłanne

Algorytm Prima:

- Algorytm rozpoczyna od wierzchołka startowego r
- W każdym kroku algorytmu zachłannie wybieramy najlżejszą krawędź dołączającą do drzewa nowy wierzchołek
- Aby łatwo wybierać najlżejsze krawędzie, wykorzystujemy kolejkę priorytetową Q typu min zawierającą wierzchołki grafu. Każdy wierzchołek v ma atrybut $v.key$ (klucz), określający minimalny koszt przyłączenia wierzchołka v do bieżącego drzewa. Porównywanie elementów kolejki Q polega na porównywaniu ich kluczy
- Zmniejszenie wartości $v.key$ oznacza zatem wywołanie na kolejce Q procedury DECREASE-KEY
- Każdy wierzchołek v posiada także atrybut $v.prev$, oznaczający wierzchołek, do którego przyłączany jest w drzewie wierzchołek v
- Minimalne drzewo rozpinające tworzą krawędzie $(v, v.prev)$, gdzie $v \in V \setminus \{r\}$

Algorytmy zachłanne

```
procedure MST-PRIM(G,w,r)
 for each u ∈ V
  u.key = ∞
  u.prev = NIL
 r.key = 0
 Q = V
 while Q ≠ ϕ
  u = EXTRACT-MIN(Q)
  for each v ∈ Adj[u]
   if v ∈ Q and w(u,v) < v.key
    v.prev = u
    v.key = w(u,v)
```

Algorytmy zachłanne

Kod Huffmana:

- Służy do kompresji danych, zwykle pozwala na zaoszczędzenie od 20 do 90% pamięci
- Jest kodem binarnym o zmiennej długości: każdy znak jest reprezentowany przez pewien ciąg bitów, długości ciągów reprezentujących różne znaki mogą być różne
- Jest kodem prefiksowym: kod żadnego znaku nie jest prefiksem (początkiem) kodu żadnego innego znaku
- Jest tworzony dla danego zbioru znaków: w wyniku każdemu znakowi z przypisana jest liczba z, jego wystąpień w kompresowanym tekście
- Jest kodem optymalnym: pozwala uzyskać maksymalny stopień kompresji

Algorytmy zachłanne

Kod Huffmana:

- Służy do kompresji danych, zwykle pozwala na zaoszczędzenie od 20 do 90% pamięci
- Jest kodem binarnym o zmiennej długości: każdy znak jest reprezentowany przez pewien ciąg bitów, długości ciągów reprezentujących różne znaki mogą być różne
- Jest kodem prefiksowym: kod żadnego znaku nie jest prefiksem (początkiem) kodu żadnego innego znaku
- Jest tworzony dla danego zbioru znaków C, w którym każdemu znakowi c przypisana jest liczba $c.f$ jego wystąpień w kompresowanym tekście
- Jest kodem optymalnym: pozwala uzyskać maksymalny stopień kompresji

Algorytmy zachłanne

Kod Huffmana:

- Służy do kompresji danych, zwykle pozwala na zaoszczędzenie od 20 do 90% pamięci
- Jest kodem binarnym o zmiennej długości: każdy znak jest reprezentowany przez pewien ciąg bitów, długości ciągów reprezentujących różne znaki mogą być różne
- Jest kodem prefiksowym: kod żadnego znaku nie jest prefiksem (początkiem) kodu żadnego innego znaku
- Jest tworzony dla danego zbioru znaków C, w którym każdemu znakowi c przypisana jest liczba $c.f$ jego wystąpień w kompresowanym tekście
- Jest kodem optymalnym: pozwala uzyskać maksymalny stopień kompresji

Algorytmy zachłanne

Kod Huffmana:

- Służy do kompresji danych, zwykle pozwala na zaoszczędzenie od 20 do 90% pamięci
- Jest kodem binarnym o zmiennej długości: każdy znak jest reprezentowany przez pewien ciąg bitów, długości ciągów reprezentujących różne znaki mogą być różne
- Jest kodem prefiksowym: kod żadnego znaku nie jest prefiksem (początkiem) kodu żadnego innego znaku
- Jest tworzony dla danego zbioru znaków C, w którym każdemu znakowi c przypisana jest liczba $c.f$ jego wystąpień w kompresowanym tekście
- Jest kodem optymalnym: pozwala uzyskać maksymalny stopień kompresji

Algorytmy zachłanne

Kod Huffmana:

- Służy do kompresji danych, zwykle pozwala na zaoszczędzenie od 20 do 90% pamięci
- Jest kodem binarnym o zmiennej długości: każdy znak jest reprezentowany przez pewien ciąg bitów, długości ciągów reprezentujących różne znaki mogą być różne
- Jest kodem prefiksowym: kod żadnego znaku nie jest prefiksem (początkiem) kodu żadnego innego znaku
- Jest tworzony dla danego zbioru znaków C, w którym każdemu znakowi c przypisana jest liczba $c.f$ jego wystąpień w kompresowanym tekście
- Jest kodem optymalnym: pozwala uzyskać maksymalny stopień kompresji

Algorytmy zachłanne

Kod Huffmana:

- Służy do kompresji danych, zwykle pozwala na zaoszczędzenie od 20 do 90% pamięci
- Jest kodem binarnym o zmiennej długości: każdy znak jest reprezentowany przez pewien ciąg bitów, długości ciągów reprezentujących różne znaki mogą być różne
- Jest kodem prefiksowym: kod żadnego znaku nie jest prefiksem (początkiem) kodu żadnego innego znaku
- Jest tworzony dla danego zbioru znaków C, w którym każdemu znakowi c przypisana jest liczba $c.f$ jego wystąpień w kompresowanym tekście
- Jest kodem optymalnym: pozwala uzyskać maksymalny stopień kompresji

Algorytmy zachłanne

Algorytm Huffmana:

- Kod reprezentujemy jako regularne drzewo binarne (każdy węzeł wewnętrzny ma dwóch synów)
- Słowo bitowe traktujemy jako ścieżkę od korzenia drzewa do liścia (kodowanego znaku), przyjmując, że 0 oznacza przejście w lewo, a 1 przejście w prawo
- Znakom często występującym w tekście powinny odpowiadać krótkie kody
- Algorytm buduje drzewo odpowiadające optymalnemu kodowi w sposób wstępujący, realizując strategię zachłanną
- W każdym kroku wybierane są dwa węzły o najmniejszej liczbie wystąpień (przykoszykujemy tu kolejkę priorytetową typu min). Zostają one synami nowego węzła, którego liczba wystąpień jest równa łącznej liczbie wystąpień synów tego

Algorytmy zachłanne

Algorytm Huffmana:

- Kod reprezentujemy jako regularne drzewo binarne (każdy węzeł wewnętrzny ma dwóch synów)
- Słowo bitowe traktujemy jako ścieżkę od korzenia drzewa do liścia (kodowanego znaku), przyjmując, że 0 oznacza przejście w lewo, a 1 przejście w prawo
- Znakom często występującym w tekście powinny odpowiadać krótkie kody
- Algorytm buduje drzewo odpowiadające optymalnemu kodowi w sposób wstępujący, realizując strategię zachłanną
- W każdym kroku wybierane są dwa węzły o najmniejszej liczbie wystąpień (wykorzystujemy tu kolejkę priorytetową Q typu min). Zostają one synami nowego węzła, którego liczba wystąpień jest równa łącznej liczbie wystąpień obu synów

Algorytmy zachłanne

Algorytm Huffmana:

- Kod reprezentujemy jako regularne drzewo binarne (każdy węzeł wewnętrzny ma dwóch synów)
- Słowo bitowe traktujemy jako ścieżkę od korzenia drzewa do liścia (kodowanego znaku), przyjmując, że 0 oznacza przejście w lewo, a 1 przejście w prawo
- Znakom często występującym w tekście powinny odpowiadać krótkie kody
- Algorytm buduje drzewo odpowiadające optymalnemu kodowi w sposób wstępujący, realizując strategię zachłanną
- W każdym kroku wybierane są dwa węzły o najmniejszej liczbie wystąpień (wykorzystujemy tu kolejkę priorytetową Q typu min). Zostają one synami nowego węzła, którego liczba wystąpień jest równa łącznej liczbie wystąpień obu synów

Algorytmy zachłanne

Algorytm Huffmana:

- Kod reprezentujemy jako regularne drzewo binarne (każdy węzeł wewnętrzny ma dwóch synów)
- Słowo bitowe traktujemy jako ścieżkę od korzenia drzewa do liścia (kodowanego znaku), przyjmując, że 0 oznacza przejście w lewo, a 1 przejście w prawo
- Znakom często występującym w tekście powinny odpowiadać krótkie kody
- Algorytm buduje drzewo odpowiadające optymalnemu kodowi w sposób wstępujący, realizując strategię zachłanną
- W każdym kroku wybierane są dwa węzły o najmniejszej liczbie wystąpień (wykorzystujemy tu kolejkę priorytetową Q typu min). Zostają one synami nowego węzła, którego liczba wystąpień jest równa łącznej liczbie wystąpień obu synów

Algorytmy zachłanne

Algorytm Huffmana:

- Kod reprezentujemy jako regularne drzewo binarne (każdy węzeł wewnętrzny ma dwóch synów)
- Słowo bitowe traktujemy jako ścieżkę od korzenia drzewa do liścia (kodowanego znaku), przyjmując, że 0 oznacza przejście w lewo, a 1 przejście w prawo
- Znakom często występującym w tekście powinny odpowiadać krótkie kody
- Algorytm buduje drzewo odpowiadające optymalnemu kodowi w sposób wstępujący, realizując strategię zachłanną
- W każdym kroku wybierane są dwa węzły o najmniejszej liczbie wystąpień (wykorzystujemy tu kolejkę priorytetową Q typu min). Zostają one synami nowego węzła, którego liczba wystąpień jest równa łącznej liczbie wystąpień obu synów

Algorytmy zachłanne

Algorytm Huffmana:

- Kod reprezentujemy jako regularne drzewo binarne (każdy węzeł wewnętrzny ma dwóch synów)
- Słowo bitowe traktujemy jako ścieżkę od korzenia drzewa do liścia (kodowanego znaku), przyjmując, że 0 oznacza przejście w lewo, a 1 przejście w prawo
- Znakom często występującym w tekście powinny odpowiadać krótkie kody
- Algorytm buduje drzewo odpowiadające optymalnemu kodowi w sposób wstępujący, realizując strategię zachłanną
- W każdym kroku wybierane są dwa węzły o najmniejszej liczbie wystąpień (wykorzystujemy tu kolejkę priorytetową Q typu min). Zostają one synami nowego węzła, którego liczba wystąpień jest równa łącznej liczbie wystąpień obu synów

Algorytmy zachłanne

```
procedure HUFFMAN(C)
 n = |C|
 Q = C
 for i = 1 to n - 1
  stwórz węzeł z
  z.left = EXTRACT-MIN(Q)
  z.right = EXTRACT-MIN(Q)
  z. = z.left.f + z.right.f
  INSERT(Q,z)
 return EXTRACT_MIN(Q)
```