Problem abstrakcyjny

Problem abstrakcyjny Q – relacja na zbiorze I *egzemplarzy* problemu i zbiorze S *rozwiązań* problemu.
Np. SHORTEST-PATH – problem najkrótszej ścieżki w grafie między parą wierzchołków:

- egzemplarz – trójka: graf, dwa wierzchołki
- rozwiązanie – ścieżka – ciąg wierzchołków (ciąg pusty – brak ścieżki)
- problem – relacja wiążąca każdy egzemplarz z rozwiązaniem – najkrótszą ścieżką w danym grafie między daną parą wierzchołków.

Problemy decyzyjne / optymalizacyjne

Problem decyzyjny – rozwiązanie stanowi odpowiedź: NIE lub TAK ($\{0, 1\}$).
Np. PATH: *Dany jest graf* $G = (V, E)$, *dwa wierzchołki* $u, w \in V$ *i nieujemna liczba całkowita* k. *Czy istnieje w* G *ścieżka od* u *do* v *o długości* $\leq k$*?*
Jeśli $i = \langle G, u, v, k \rangle$ jest egzemplarzem, to PATH$(i) = 1$, jeśli najkrótsza ścieżka od u do v w G ma długość $\leq k$, a w p.p. PATH$(i) = 0$.
Problem optymalizacyjny – znaleźć rozwiązanie o najmniejszej lub największej wartości pewnego parametru (np. ścieżkę o najmniejszej długości).
Algorytm rozwiązujący problem optymalizacyjny można wykorzystać do rozwiązania odpowiedniego problemu decyzyjnego (np. znaleźć najkrótszą ścieżkę i sprawdzić czy jest dłuższa niż k.)

Kodowanie

Egzemplarze problemu muszą być kodowane w sposób zrozumiały dla programu – kodowanie binarne.
Liczby naturalne kodujemy w systemie o ustalonej podstawie > 1. (długość reprezentacji – logarytmiczna wzgl. wartości liczby) Cyfry – n.p. w kodzie ASCII.
Można również ustalić "zwarte" sposoby kodowania obiektów złożonych (zbiory, grafy, ciągi). Np. zbiory – wyliczenie zakodowanych elementów oddzielonych kodami znaku ',' między kodami znaków '{' i '}'.
Problem konkretny – problem, którego zbiór egzemplarzy jest zakodowany. (Kod egzemplarza i oznaczamy $< i >$.)
Algorytm *rozwiązuje* konkretny problem w czasie $O(T(n))$, jeśli dla egzemplarza problemu o długości kodu n wyznacza rozwiązanie w czasie $O(T(n))$.

Problemy jako języki formalne

Alfabet: $\Sigma = \{0, 1\}$. *Język* zbiór napisów z Σ^*.
Problem decyzyjny Q identyfikujemy ze zbiorem (kodów) egzemplarzy: $L = \{x \in \Sigma^* : Q(x) = 1\}$
Np. PATH$= \{< G, u, v, k >: G = (V, E)$ jest grafem, $u, v \in V$, $k \geq 0$ jest liczbą całkowitą
oraz istn. ścieżka od u do v w G o dł. $\leq k\}$
Algorytm A *akceptuje* słowo $x \in \Sigma^*$ jeśli dla x oblicza $A(x) = 1$. A *odrzuca* x, jeśli $A(x) = 0$.
Język L jest *rozpoznawalny w czasie wielomianowym* przez A, jeśli istn. stała k, taka, że dla każdego $x \in \Sigma^*$ A akceptuje x w czasie $O(n^k)$ jeśli $x \in L$, i A nie akceptuje x jeśli $x \notin L$.
Język L jest *rozstrzygalny w czasie wielomianowym* przez A, jeśli istn. stała k, taka, że dla każdego $x \in \Sigma^*$ a akceptuje x w czasie $O(n^k)$ jeśli $x \in L$, i A odrzuca x w czasie $O(n^k)$ jeśli $x \notin L$.

Klasa złożoności

Klasa złożoności – zbiór języków L, dla których istnieje algorytm o danej złożoności rozstrzygający czy dane słowo należy do L.

Np. P$= \{L \subseteq \{0,1\}^* :$ istnieje algorytm A rozstrzygający o L w czasie wielomianowym$\}$

Tw. P$= \{L : L$ jest akceptowalny w czasie wielomianowym$\}$

D-d. Jeśli L – akceptowalny w czasie wielomianowym, to istnieje algorytm A, który każde słowo z L długości n akceptuje w czasie $\leq c \cdot n^k$, dla pewnych stałych c, k. Można skonstruować algorytm A', który w czasie wielomianowym rozstrzyga czy słowo x należy do L: A' wyznacza długość n słowa x a następnie "symuluje i mierzy czas działania" algorytmu A. Jeśli w czasie cn^k A nie zaakceptuje x, to A' odrzuca x. $\square$

Algorytmy weryfikacji

Przykład:
HAM-CYCLE$= \{< G >: G$ jest grafem hamiltonowskim$\}$
Graf $G = (V, E)$ jest *hamiltonowski*, jeśli zawiera cykl (*cykl Hamiltona*), w którym każdy wierzchołek z V występuje dokładnie raz.
Nie jest znany algorytm, który w czasie wielomianowym rozstrzyga, czy graf jest hamiltonowski.
Natomiast łatwo skonstruować algorytm wielomianowy, który dla danego grafu G i cyklu c, sprawdza czy c jest cyklem Hamiltona w G.
Algorytm weryfikacji – algorytm dwuparametrowy A. A *weryfikuje* x, jeśli istnieje y (*świadectwo*), takie, że $A(x, y) = 1$. *Język weryfikowany* przez A: $L = \{x \in \{0,1\}^* : \exists_{y \in \{0,1\}^*} A(x, y) = 1\}$

Klasa NP

L należy do klasy NP (*nondeterministic polynomial time*) jeśli istnieje dwuparametrowy wielomianowy algorytm A i stała c takie, że:
$L = \{x \in \{0,1\}^* : \exists_{y \in \{0,1\}^*} \ |y| = O(|x|^c) \wedge A(x,y) = 1\}$
Niech $\bar{L} = \{0,1\}^* \setminus L$. Wtedy co-NP$= \{L \subseteq \{0,1\}^* : \bar{L} \in$ NP$\}$.
Ponieważ $L \in$P implikuje $\bar{L} \in$P oraz P$\subseteq$NP, więc P$\subseteq$NP$\cap$co-NP. Zachodzi jedna z możliwości:

- P$=$NP$=$co-NP
- P$\subsetneq$NP$=$co-NP
- P$=$NP$\cap$co-NP oraz NP$\neq$co-NP
- P$\subsetneq$NP$\cap$co-NP oraz NP$\neq$co-NP

Redukowalność

Język L_1 jest *redukowalny w czasie wielomianowym* do L_2 ($L_1 \leq_P L_2$), jeśli istnieje obliczalna w czasie wielomianowym funkcja $f : \{0,1\}^* \to \{0,1\}^*$ taka, że

$$(x \in L_1) \equiv (f(x) \in L_2)$$

f – *funkcja redukcji*, algorytm F liczący f – *algorytm redukcji*.

Lemat. Jeśli $L_1 \leq_P L_2$ i $L_2 \in$P, to $L_1 \in$P.

D-d. Czas działania algorytmu liczącego $f(x)$, a zatem i rozmiar wyniku jest wielomianowy wzgl. $|x|$. Stąd wielomianowy algorytm A_2 rozstrzygający o L_2, uruchomiony na $f(x)$ też działa w czasie wielomianowym wzgl. $|x|$. (*Suma i złożenie wielomianów jest wielomianem.*) $\square$

NP-zupełność

Język L jest NP-trudny, jeśli $L' \leq_P L$ dla każdego $L' \in$NP.
Język L jest NP-zupełny, jeśli L jest NP-trudny oraz $L \in$NP.
NPC – klasa języków NP-zupełnych.
Tw. Jeśli jakiś problem NP-zupełny jest w P, to P=NP. Jeśli jakikolwiek problem z NP nie jest w P, to żaden problem NP-zupełny nie jest w P.
D-d. Jeśli $L \in$P i $L \in$NPC, to dla każdego $L' \in$NP zachodzi $L' \leq_P L$ i (z Lematu) $L' \in P$.
Niech $L \in$NP\P i $L' \in$NPC$\cap$P. Wtedy $L \leq_P L'$ i (z Lematu) $L \in$P. Sprzeczność. $\square$

Problem SAT

Egzemplarz problemu SAT: formuła logiczna ϕ złożona ze zmiennych $x_1, x_2, \ldots$, i spójników logicznych $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$.
Wartościowanie – przypisanie każdej zmiennej wartości 0 albo 1.
Wartościowanie spełniające – takie wartościowanie, że ϕ przybiera wartość 1.
Jeśli dla ϕ istnieje wartościowanie spełniające, to ϕ jest *spełnialna*.
SAT$= \{< \phi >: \phi$ jest spełnialną formułą logiczną$\}$
Tw. SAT jest NP-zupełny.
D-d. Dowód – na wykładzie ze złożoności obliczeniowej. $\square$
Lemat. Jeśli $L' \leq_P L$, dla pewnego $L' \in$NPC, to L jest NP-trudny. Jeśli dodatkowo $L \in$NP, to $L \in$ NPC. $\square$

3-CNF-SAT

Formuła jest w `koniunkcyjnej postaci normalnej` (CNF) jeśli jest zapisana jako koniunkcja klauzul, z których każda jest alternatywą jednego lub więcej literałów (*literał* – zmienna lub negacja zmiennej).
Formuła jest w postaci 3-CNF, jeśli jest w postaci CNF i każda klauzula zawiera dokładnie 3 literały.
3-CNF-SAT jest zbiorem spełnialnych formuł w postaci 3-CNF.

Tw. 3-CNF-SAT jest problemem `NP-zupełnym`.

D-d. Dowód – na *Złożoności Obliczeniowej*. (*Dla danego wartościowania łatwo zweryfikować w czasie wielomianowym, że spełnia formułę 3-CNF, czyli* 3-CNF-SAT$\in NP$. *Ponadto pokazuje się jak w czasie wielomianowym przekształcić dowolną formułę logiczną w formułę 3-CNF, czyli – że* SAT$\leq_P$3-CNF-SAT.) $\square$

Problem kliki

Klika w grafie $G = (V, E)$ – podzbiór $V' \subseteq V$ taki, że każda para wierzchołków z V' jest połączona krawędzią.
CLIQUE=$\{< G, k >$: G – graf zawierający klikę rozmiaru $k\}$
Tw. CLIQUE jest problemem NP-zupełnym.
D-d CLIQUE$\in$NP – mając dane V' łatwo zweryfikować w czasie wielomianowym czy V' jest kliką w G i $|V'| = k$.
3-CNF-SAT$\leq_P$CLIQUE: Niech $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ – formuła 3-CNF, o k klauzulach, gdzie $C_r = (l_1^r \vee l_2^r \vee l_3^r)$. Konstruujemy graf G, gdzie każdej C_r odpowiada grupa trzech wierzchołków v_1^r, v_2^r, v_3^r. Wierzchołki v_i^r, v_j^s łączymy krawędzią jeśli:

- $r \neq s$ (v_i^r, v_j^s są w różnych grupach), oraz
- l_i^r nie jest negacją l_j^s

...

Problem kliki

Jeśli G zawiera klikę V' rozmiaru k, to każda grupa C_r jest reprezentowana w V' przez jeden wierzchołek $v^r_{i_r}$. Żadne dwa wierzchołki w V' nie odpowiadają sprzecznym literałom. Stąd istnieje wartościowanie (przypisujące 1 literałom odp. wierzchołkom V'), takie że ϕ jest spełniona.
Z drugiej strony: jeśli ϕ jest spełnialne, to wartościowanie spełniające ϕ przypisuje 1 k literałom w różnych grupach. Literały te są niesprzeczne, więc stanowią klikę rozmiaru k w G.
Konstrukcja G z ϕ – w czasie wielomianowym. $\square$

Pokrycie wierzchołkowe

Pokrycie wierzchołkowe grafu $G =< V, E >$ – zbiór $V' \subseteq V$, taki, że jeśli $(u, v) \in E$, to $u \in V'$ lub $v \in V'$.

VERTEX-COVER$= \{< G, k >: G$ – graf zawierający pokrycie wierzchołkowe rozmiaru $k\}$

Tw. VERTEX-COVER jest NP-zupełny.

D-d. VERTEX-COVER$\in$NP: Łatwo zweryfikować w czasie wielomianowym, że dany zbiór wierzchołków jest pokryciem danego grafu rozmiaru k.

CLIQUE$\leq_P$VERTEX-COVER: Redukcja w czasie wielomianowym: przekształcamy dane $< G, k >$ na $< \bar{G}, |V| - k >$, gdzie $\bar{G} = (V, \bar{E})$ *dopełnienie* grafu $G = (V, E)$ (t.j. $\bar{E} = V \times V \setminus E$).

Fakt: G ma klikę rozmiaru k wtedy i tylko wtedy gdy $\bar{G}$ ma pokrycie wierzchołkowe rozmiaru $|V| - k$.

...

Pokrycie wierzchołkowe

D-d Faktu:
Niech G zawiera klikę V', $|V'| = k$. Niech $(u, v) \in \bar{E}$, czyli $(u, v) \notin E$. Stąd $u \notin V'$ lub $v \notin V'$, czyli u lub v jest w $V \setminus V'$. Zatem $V \setminus V'$ jest pokryciem (rozmiaru $|V| - k$) w $\bar{G}$.
Niech $\bar{G}$ zawiera pokrycie V' rozmiaru $|V| - k$. Jeśli $(u, v) \in \bar{E}$, to u lub v jest w V'. Równoważnie: Jeśli $u \notin V'$ i $v \notin V'$, to $(u, v) \notin \bar{E}$. Inaczej: Jeśli $u \in V \setminus V'$ i $v \in V \setminus V'$, to $(u, v) \in E$. $V \setminus V'$ jest kliką (rozmiaru k) w G. $\square$

Problem sumy podzbioru

SUBSET-SUM$= \{< S, t >$: istnieje $S' \subseteq S$ taki, że $t = \sum_{s \in S'} s\}$

Tw. SUBSET-SUM jest NP-zupełny.

D-d. SUBSET-SUM$\in$NP: Można w czasie wielomianowym zweryfikować, że $S' \subseteq S$ oraz $t = \sum_{s \in S'} s$.

VERTEX-COVER$\leq_P$SUBSET-SUM: Przekształcamy $< G, k >$ w $< S, t >$. Niech $G = (V, E)$, gdzie $V = \{v_0, \ldots, v_{n-1}\}$ i $E = \{e_0, \ldots, e_{m-1}\}$. Tworzymy $t = k4^m + \sum_{j=0}^{m-1} 2 \cdot 4^j$ i zbiór $S = \{x_0, \ldots, x_{n-1}\} \cup \{y_0, \ldots, y_{m-1}\}$, gdzie $y_j = 4^j$, $x_i = 4^m + \sum_{j=0}^{m-1} b_{ij} 4^j$, dla

$$b_{ij} = \begin{cases} 1, & \text{jeśli } v_i \text{ jest końcem } e_j \\ 0, & \text{w przeciwnym razie} \end{cases}$$

Problem sumy podzbioru

Fakt: $< G, k > \in$ VERTEX-COVER $\equiv < S, t > \in$ SUBSET-SUM.
Załóżmy, że $V' = \{v_{i_1}, \ldots, v_{i_k}\}$ – pokrycie wierzchołkowe G i $|V'| = k$. Niech $X' = \{x_{i_1}, \ldots, x_{i_k}\}$, $Y' = \{y_j$:tylko jeden koniec e_j należy do $V'\}$ i $S' = X' \cup Y'$. Wtedy: $\sum_{s \in S'} s = t$.
(*Dodajemy liczby z* S' *w systemie czwórkowym: każdej pozycji* j, $0 \leq j \leq m-1$ *odpowiadają dwie jedynki:*

- *obie w liczbach z* X' *jeśli oba końce* e_j *są w* V'
- *jedna z* X' *i jedna z* Y' *jeśli jeden koniec* e_j *jest w* V'

Zatem na pozycjach $0, \ldots, m-1$ *nie ma przeniesień i są same dwójki. Na pozycjach wyższych* $\geq m$ *jest zapisana liczba* $|X'| = |V'| = k$. *Zatem uzyskujemy zapis* t *z systemie czwórkowym.*)

...

Problem sumy podzbioru

Załóżmy, że w S isnieje podzbiór S' o sumie t. Niech $S' = \{x_{i_1}, \ldots, x_{i_r}\} \cup \{y_{j_1}, \ldots, y_{j_p}\}$. Ponownie przyjmujemy system czwórkowy. Zauważmy, że dla $0 \leq j \leq m-1$ w S' mogą być co najwyżej 3 liczby z jedynką na pozycji j: dwie typu x_* odpowiadające końcom e_j i jedna y_j. Zatem na pozycjach 0,. . . ,$m-1$ nie ma przeniesień. Ponieważ t ma w zapisie czwórkowym na pozycji j cyfry '2', więc w S' muszą być 2 liczby z jedynką na pozycji j z czego $\geq$ jedna jest typu x_*. Zatem $\{v_{i_1}, \ldots, v_{i_r}\}$ jest pokryciem G.
Ponieważ na pozycjach $\geq m$ liczby t jest ilość dodanych liczb typu x_*, więc $r = k$.
□

Problem cyklu Hamiltona

HAM-CYCLE$= \{< G >:$ G jest grafem hamiltonowskim$\}$

Tw. HAM-CYCLE jest NP-zupełny

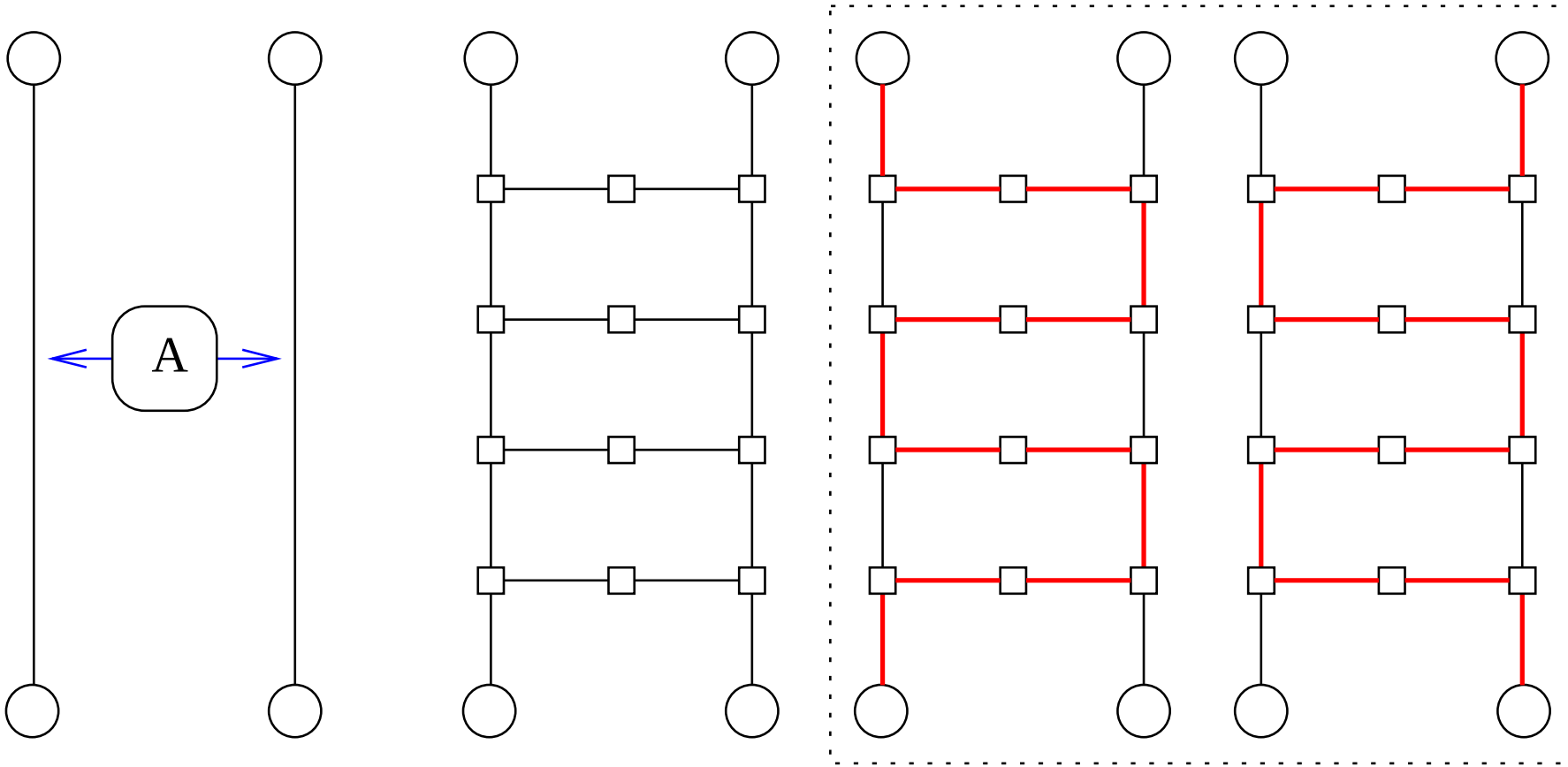
D-d. HAM-CYCLE$\in$NP: Dla danego grafu G i ciągu wierzchołków c można zweryfikować w czasie wielomianowym, że c jest cyklem Hamiltona grafu G.

Pokażemy, że można zredukować w czasie wielomianowym 3-CNF-SAT do HAM-CYCLE.

Dla danego wyrażenia w postaci 3-CNF konstruujemy graf, który zawiera cykl Hamiltona wtedy i tylko wtedy gdy to wyrażenie jest spełnialne.

Gadżet A

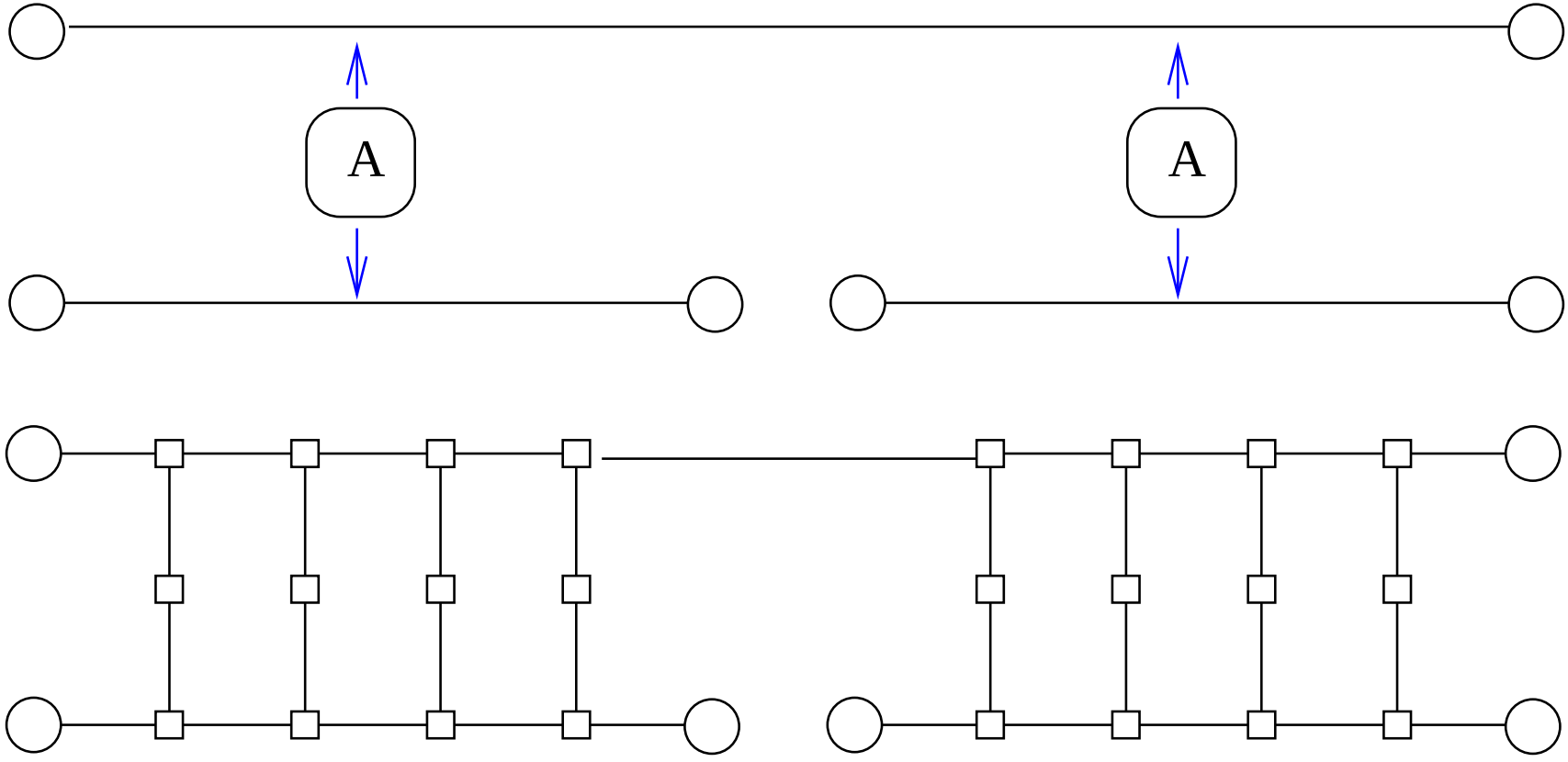
Gadżet A dla między krawędziami.



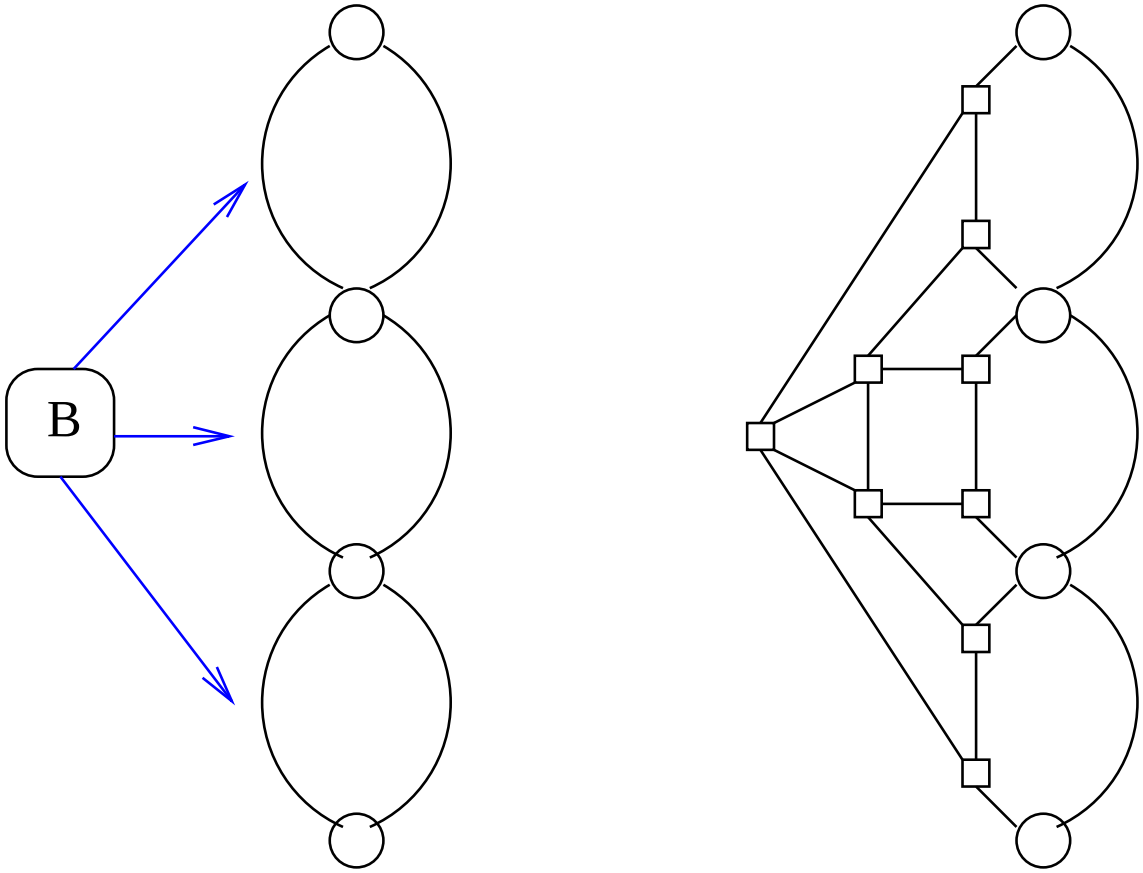
Z resztą grafu są połączone tylko okrągłe wierzchołki.

Gadżet A

Połączenie jednej krawędzi wieloma gadżetami A z innymi krawędziami:

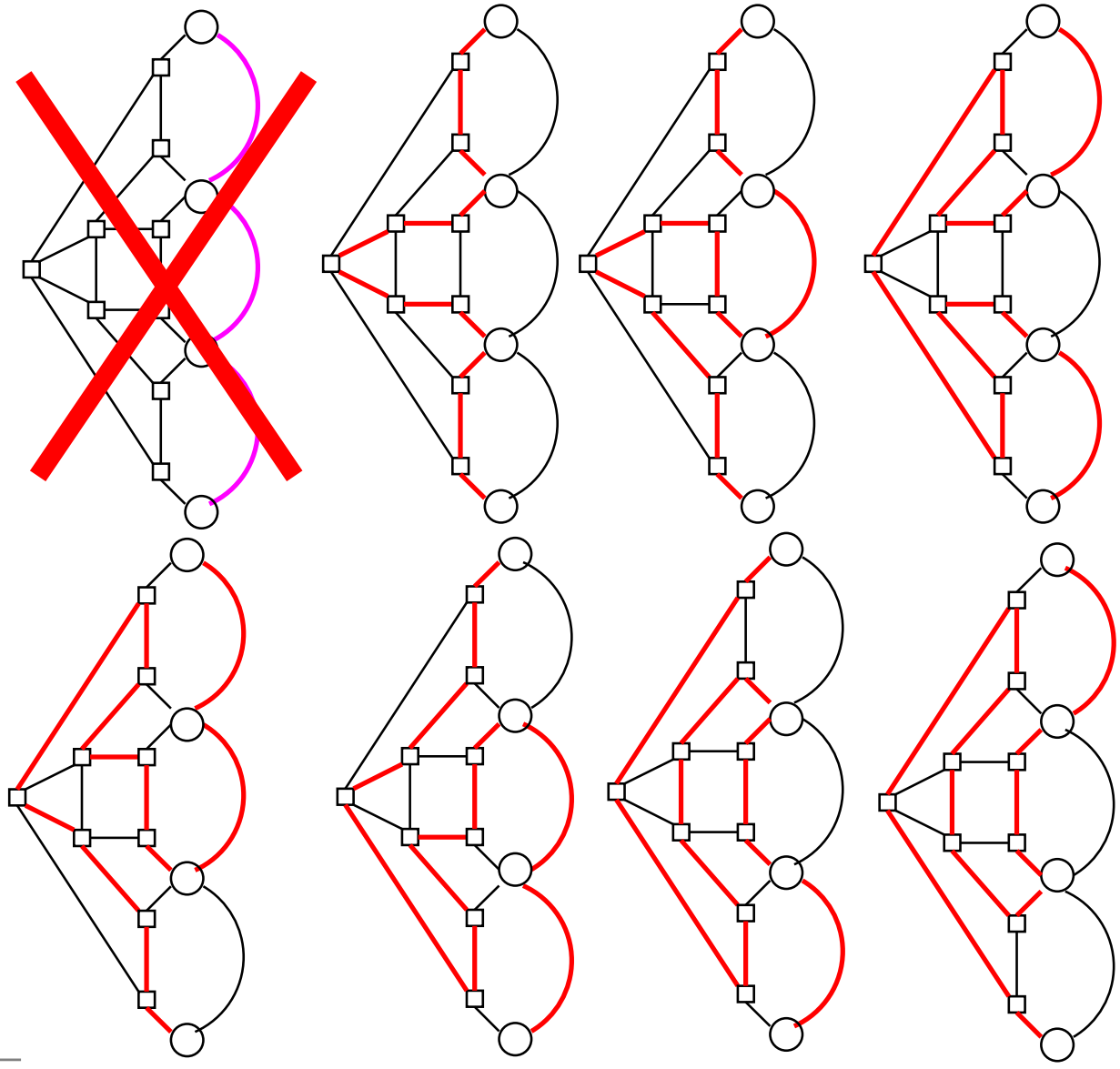


Gadżet B



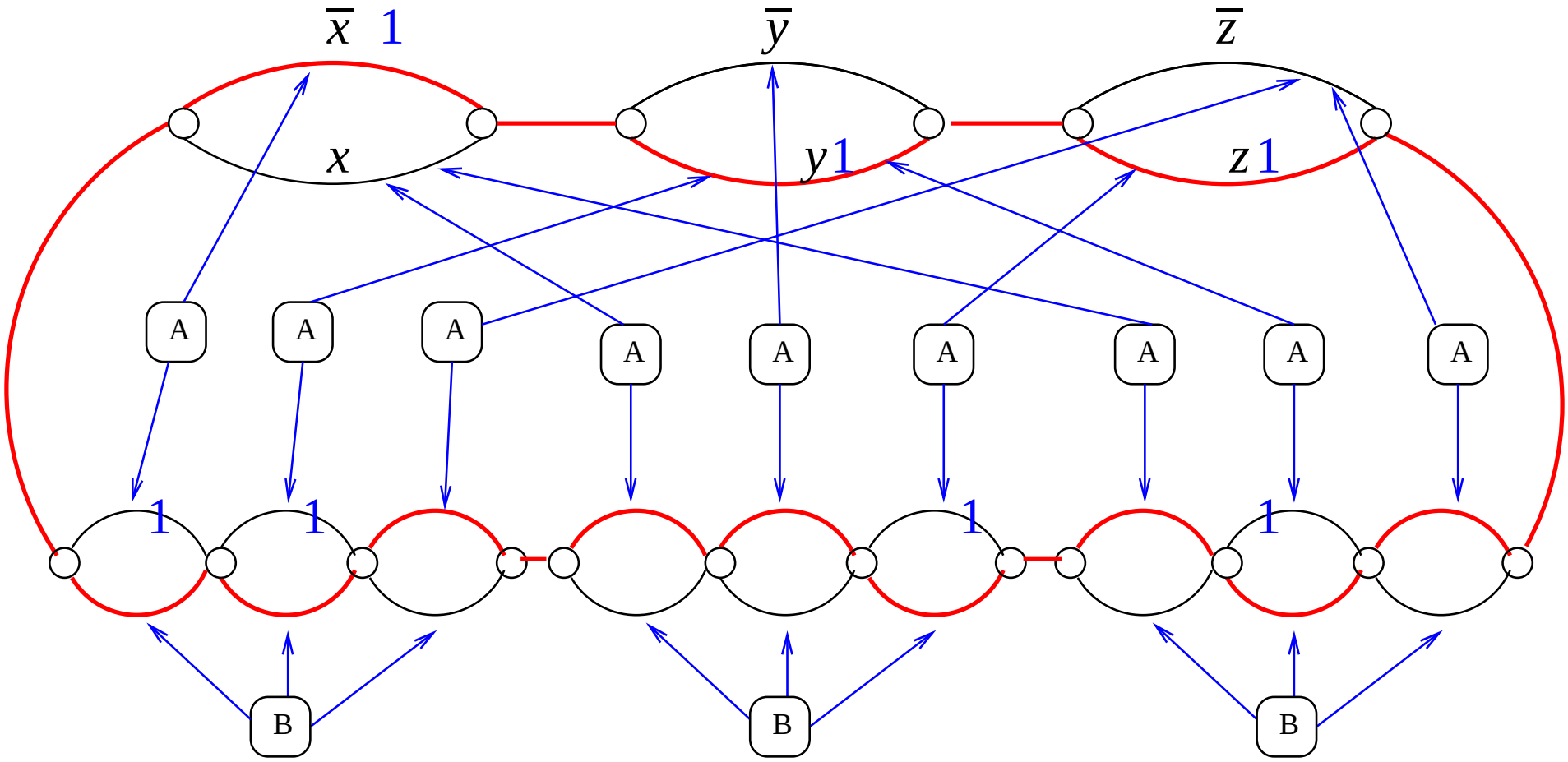
Tylko górny i dolny wierzchołek połączony z resztą grafu. Cykl Hamiltona może zawierać dowolny podzbiór lewych (*wolnych*) krawędzi z wyjątkiem wszystkich trzech.

Gadżet B



Redukcja

Przykład: $(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee \bar{z})$



Redukcja

W ogólnym przypadku:

1. Dla każdej zmiennej x_i tworzymy podgraf z dwóch wierzchołków, połączonych parą krawędzi (dolna krawędź – etykieta x_i, górna krawędź – etykieta $\bar{x}_i$). Łączymy je jak w przykładzie (w górnym rzędzie).
2. Dla każdej klauzuli C_j tworzymy gadżet B B_j. Łączymy je jak w przykładzie (w dolnym rzędzie).
   - Jeśli na k-tej pozycji C_j jest x_i, to wstawiamy gadżet A między k-tą *wolną* krawędzią gadżetu B_j a krawędzią o etykiecie x_i.
   - Jeśli na k-tej pozycji C_j jest $\bar{x}_i$, to wstawiamy gadżet A między k-tą *wolną* krawędzią gadżetu B_j a krawędzią o etykiecie $\bar{x}_i$.
3. Łączymy górny i dolny rząd jak w przykładzie.

Redukcja

Fakt: W tak skonstruowanym grafie jest cykl Hamiltona wtedy i tylko wtedy gdy formuła jest spełnialna.

Jeśli w grafie jest cykl Hamiltona c, to wyznacza wartościowanie spełniające formułę: Jeśli w tym cyklu "użyta" była krawędź o etykiecie x_i to wartościujemy $x_i = 1$. W przeciwnym wypadku była "użyta" krawędź $\bar{x}_i$ i wartościujemy $x_i = 0$.

Weźmy gadżet B dla klauzuli C_j. Cykl c "omija" jedną z *wolnych* krawędzi B. c musi zatem "użyć" krawędzi połączonej z nią gadżetem A – czyli literał C_j odpowiadający tej *wolnej* krawędzi ma wartość 1 i C_j jest spełniona. To samo zachodzi dla pozostałych klauzul.

Redukcja

Jeśli istnieje wartościowanie s spełniające formułę, to wyznacza ono cykl Hamiltona: Jeśli w s $x_i = 1$ to *wybieramy* krawędź x_i. W przeciwnym wypadku – *wybieramy* $\bar{x}_i$.
Ponieważ s – spełniające, więc dla każdej klauzuli C_j przynajmniej jedna z *wolnych* krawędzi jej gadżetu B jest połączona gadżetem A z *wybraną* krawędzią. Można zatem poprowadzić cykl Hamiltona, który "omija" te *wolne* krawędzie i "używa" *wybrane*.
Redukcja formuły 3-CNF do grafu – w czasie wielomianowym (Gadżety stałych rozmiarów; po jednym gadżecie B na klauzulę i po jednym gadżecie A na literał klauzuli.)
□

Problem Komiwojażera

TSP=$\{< G, c, k >: G = (V, V \times V)$ – graf pełny,
c – f-cja kosztu $V \times V \to Z$,
$k \in Z$, oraz w G istnieje cykl Hamiltona o koszcie $\leq k\}$

Tw. TSP jest NP-zupełny.

D-d. TSP$\in$NP: Dla danego ciągu wierzchołków, łatwo sprawdzić w czasie wielomianowym czy jest cyklem Hamiltona o koszcie $\leq k$.

HAM-CYCLE$\leq_P$TSP: Dla danego grafu $G = (V, E)$ tworzymy funkcję kosztu:

$$c(i,j) = \begin{cases} 0, & \text{jeśli } (i,j) \in E \\ 1, & \text{jeśli } (i,j) \notin E \end{cases}$$

Fakt: W G jest cykl Hamiltona $\equiv < (V, V \times V), c, 0 > \in$TSP.

□